НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему: «Виявлення захворювань за зображенням очного дна з використанням нейронних мереж»

Виконав:
Студент IV курсу, групи КА-92
Яковлєв Сергій Олександрович _____

Керівник:
доцент, к.ф.-м.н. Каніовська І.Ю. _____

Консультант з економічного розділу:
доцент, к.е.н. Рощина Н.В. _____

Консультант з нормоконтролю:
доцент, к.ф.-м.н. Статкевич В.М. _____

Рецензент:
доцент, к.ф.-м.н. Буценко Ю.П. _____

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

Київ - 2023 року

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

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2023 р.

ЗАВДАННЯ

на дипломну роботу студента

Яковлєва Сергія Олександровича

1. Тема роботи «Виявлення захворювань за зображенням очного дна з використанням нейронних мереж», керівник роботи доцент, к. ф.-м. н. Каніовська І.Ю., затверджені наказом по університету від «30» травня 2023 р. № 2065-с

2. Термін подання студентом роботи __.06.2023.

3. Вихідні дані до роботи: модель класифікації зображень, модель виявлення деяких захворювань очей за зображенням очного дна.

4. Зміст роботи: Застосування нейронних мереж для виявлення захворювань очей, за знімком очного дна, на прикладі двох найпоширеніших з них – катаракти та діабетичної ретинопатії.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., доцент		

7. Дата видачі завдання:_____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Затвердження теми ДР	24.04.2023-30.04.2023	виконано
2	Ознайомлення зі структурою БДР згідно з Положенням про державну атестацію студентів НТУУ «КПІ ім. І. Сікорського»	17.04.2023-23.04.2023	виконано
3	Ознайомлення з ДСТУ 3008-95 та стандарти ЄСПД	17.04.2023-23.04.2023	виконано
4	Проведення дослідження за темою БДР під керівництвом керівника	24.04.2023-30.04.2023	виконано
5	Завершення роботи над першим варіантом частини БДР	01.05.2023-15.05.2023	виконано
6	Проведення роботи над експериментальною частиною БДР	15.05.2023-21.05.2023	виконано
7	Проведення роботи над програмним продуктом	22.05.2023-28.05.2023	виконано
8	Оформлення БДР та аналіз отриманих результатів	22.05.2023-28.05.2023	виконано

Студент Сергій ЯКОВЛЄВ

Керівник Ірина КАНІОВСЬКА

РЕФЕРАТ

Дипломна робота містить 116 сторінок текстової частини, 45 ілюстрацій, 13 таблиць, 2 додатки, 22 бібліографічних посилання.

Ключові слова: НЕЙРОННІ МЕРЕЖІ, КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ, ЗОБРАЖЕННЯ ОЧНОГО ДНА, ЗАХВОРЮВАННЯ ОЧЕЙ, КАТАРАКТА, ДІАБЕТИЧНА РЕТИНОПАТІЯ.

Об’єкт дослідження: розпізнавання деяких захворювань очей за зображенням очного дна.

Мета дослідження: проаналізувати існуючі методи класифікації зображень, побудувати на їх основі моделі для виявлення катаракти та діабетичної ретинопатії у випадку, коли в якості альтернативи до них виступають знімки не тільки здорового очного, але і знімки з іншими захворюваннями, відмінними від них, можливо – кількома.

Використані моделі: в якості базових моделей було використано моделі згорткових нейронних мереж, що базуються на VGG-16, VGG-19, ResNet50.

Отримані результати: побудовано дві моделі нейронних мереж для задачі бінарної класифікації – одна для виявлення катаракти, інша – для виявлення діабетичної ретинопатії, які можуть виявляти відповідні захворювання з, за великим рахунком, прийнятними значеннями метрик якості.

В рамках подальшого дослідження пропонується побудувати аналогічним чином моделі для виявлення менш поширених захворювань очей, зокрема у випадку, коли перед класифікацією доцільно виокремити певну частину зображення очного дна, за якою виявляти відповідне захворювання буде простіше.

ABSTRACT

The diploma thesis contains 116 pages of the text part, 45 illustrations, 13 tables, 2 appendices, 22 references.

Keywords: NEURAL NETWORKS, IMAGE CLASSIFICATION, FUNDUS IMAGE, OCULAR DISEASE, CATARACT, DIABETIC RETINOPATHY.

Object of study: detection of some ocular diseases by fundus image.

Purpose: to analyze existing methods for image classification, build one their base models for detecting cataract and diabetic retinopathy in case, when as alternatives to them goes not only images of healthy fundus, but also images of other diseases, different from corresponding.

Used models: as base models were used models of convolutional neural networks, based on VGG-16, VGG-19, ResNet50.

Results: we built two neural network models for binary classification task – one for cataract detection, another one – for diabetic retinopathy detection, which are capable of detecting corresponding diseases with, generally saying, acceptable values of quality metrics.

As part of further research, it is proposed to build in the analogic way models for detecting less widespread ocular diseases, including case, where before classifying image is might be useful to localize and differentiate certain part of fundus image, which will be simpler for detecting corresponding disease.

ЗМІСТ

ВСТУП

На сьогоднішній день, нейронні мережі вже суттєво посунули традиційні алгоритми машинного навчання у розв’язанні задач і знайшли своє призначення в багатьох галузях людського життя – освіта, банківська справа, сфера обслуговування, індустрія розваг, тощо, зокрема – медицина.

Можна навести багато прикладів того, як нейронні мережі можуть бути корисними у сфері охорони здоров’я. Наприклад, їх можна використовувати для виявлення захворювань за даними обстеження пацієнта – виявлення захворювань головного мозку за знімками магнітної резонансної томографії, виявлення переломів та захворювань внутрішніх органів за рентгенівськими знімками, виявлення захворювань за загальним аналізом крові, тощо. Зокрема, можна виділити захворювання очей.

Ні для кого не секрет, що зір – є одним з найважливіших почуттів людини, втрата якого могла би стати суттєвою перепоною у подальшому житті людини. При цьому, для втрати зору зовсім не обов’язково зазнавати фізичних пошкоджень ока, як то потрапляння в око сторонніх предметів. Є доволі великий список захворювань, які у випадку відсутності необхідного нагляду можуть привести то часткової чи повної втрати зору. Тому такі речі треба вміти діагностувати вчасно і якісно. При цьому, науково доведено, що більшість цих захворювань мають характерні ознаки, яку кваліфікований спеціаліст-офтальмолог здатен побачити на знімку очного дна або передньої частини ока пацієнта.

Метою даного дослідження є розробка програмного продукту, що за завантаженим зображенням очного дна людини, міг би виявити, є в неї якісь захворювання, чи нема.

В першому розділі буде розглянуто актуальність даної проблеми, певні теоретичні відомості щодо основних захворювань, а також певні методи, що сьогодні застосовуються для розв’язання даної задачі.

В другому розділі буде розглянуто увесь необхідний інструментарій щодо машинного навчання, в абсолютній більшості - нейронних мереж, необхідний для розробки програмного продукту.

В третьому розділі буде розглянуто набір даних для навчання, виконано його обробку та проведено експерименти з ними у вигляді побудови різних моделей нейронних мереж, навчених на ньому.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

В наш час захворювання очей є доволі поширеною проблемою. Згідно даних Всесвітньої організації охорони здоров'я [1], зараз у світі принаймні 1 мільярд осіб, які мають проблеми із зором в сенсі короткозорості, або далекозорості, при чому ці проблеми можуть вражати людей у будь-якому віці, можуть привести до повної втрати зору людей і мати для їх життя наслідки різного характеру в довгостроковій перспективі. Так, наприклад, діти із серйозними порушеннями зору можуть поступатись в академічних досягненнях, а дорослі – в робочій продуктивності.

Можна посилатись в певній мірі на те, що у наш час, еру мобільних пристроїв та комп'ютерів, зір і загальне здоров'я погіршується через їх використання [2], але часто також на погіршення зору впливають певні захворювання, як то катаракта, діабет, тощо.

Говорячи про методи виявлення даних захворювань, важливим є поняття очного дна. Власне, очне дно – це видима при офтальмологічному обстеженні частина внутрішньої поверхні ока, що складається з диску зорового нерву, сітківки та судинної оболонки [3].

На практиці, професійні лікарі-офтальмологи здатні виявляти дані захворювання за зображенням очного дна людини. Приклад такого зображення наведено на рисунку 1.1.

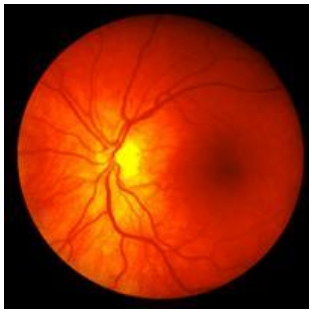


Рисунок 1.1 Приклад зображення очного дна

Актуальність даного дослідження полягає в тому, що нейронні мережі в класифікації захворювань вже неодноразова перевершали професійних лікарів, і при цьому можливість зробити знімок очного дна є доволі доступною у переважній більшості країн світу, для цього потрібна лише так звана фундус-камера [4], приклад якої зображено на рисунку 1.2.

Рисунок 1.2 Приклад фундус-камери

Тому, розробка даних алгоритмів могла би надати можливість автоматизувати процес виявлення захворювань очей, і, зокрема, прискорити цей процес, та зробити його доступним, наприклад, для використання у великих компаніях, з метою відслідковування здоров’я очей працівників, оскільки, як зазначалось раніше, проблеми із зором здатні доволі суттєво впливати на працездатність людини, що не є добре для бізнесу.

1.2 Основні захворювання

Серед найпоширеніших захворювань очей можна виділити катаракту та діабетичну ретинопатію. Розглянемо кожне з них більш детально. Зауважимо, що основну інформацію було взято з [5].

Катаракта – це помутніння кришталика ока. Вона є головною причиною втрати зору у світі і при цьому може виникнути в будь-якому віці з різних причин, або бути присутньою від народження. Приклад знімку очного дна при катаракті наведено на рисунку 1.3.

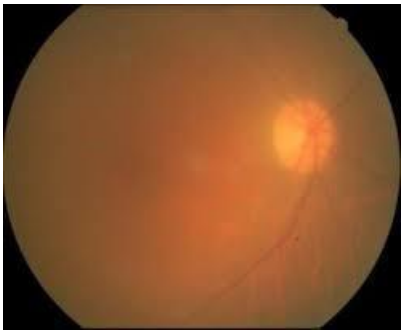


Рисунок 1.3 Очне дно при катаракті

Діабетична ретинопатія – це одне із ускладнень діабету. Дане захворювання проявляється у прогресуючому пошкодженні кровоносних судин сітківки, яка є світлочутливою тканиною в задній частині ока і необхідна для нормального зору. Приклад очного дна при діабетичній ретинопатії зображено на рисунку 1.4.

Рисунок 1.4 Очне дно при діабеті

1.3 Огляд існуючих методів розв'язання даної задачі

Розглянемо декілька підходів, що застосовуються на практиці для виявлення захворювань очей за знімком очного дня.

Частіше за все дослідники концентрують свою увагу на виявленні якогось одного конкретного захворювання, як то діабетична ретинопатія, катаракта, тощо, тим самим зводячи поставлену задачу до класифікації. Це в принципі має сенс і є доволі універсальним засобом, оскільки задачу багато-класової класифікації для кількох захворювань і випадку здорового ока тут застосовувати не доречно, оскільки вона заздалегідь вимагає виконання умови несумісності класів, тобто кожен знімок треба віднести рівно до одного класу, а на практиці дійсно можливі випадки, коли на очному дні зі знімку простежуються ознаки кількох захворювань, і при цьому, людина, якій робили цей знімок очного дна, дійсно їх має. Що правда, хоч захворювання виявляють і окремо, але часто паралельно розглядають різні стадії цього захворювання. Так, наприклад, в [6], розглядається як задача багато-класової класифікації для діабетичної ретинопатії, в якій у наборі даних один клас – відсутність захворювання, а інші 4 – різні стадії ретинопатії, так і паралельно розглядається задача бінарної класифікації, де просто перевіряється наявність/відсутність захворювання.

Отже, за мету в цій роботі можна взяти завдання побудувати моделі класифікації зображень для різних захворювань, які би на вхід отримували знімок очного дна людини, незалежно від того – лівого чи правого ока, і за ним могли би виявляти відсутність чи наявність відповідного захворювання. Щодо захворювань, ми власне візьмемо зазначені раніше та найбільш поширені серед них – катаракту та діабетичну ретинопатію.

Висновки до розділу 1

В цьому розділі було розглянуто все, що стосується актуальності даного дослідження, і з’ясовано, навіщо це все потрібно.

Також було розглянуто теоретичні відомості про основні захворювання очей – катаракту, діабетичну ретинопатію, які власне і було обрано для розв’язання задачі, і, власне, було сформульовано мету даної роботи – розробити моделі класифікації для виявлення зазначених захворювань за знімком очного дна людини, зробленим за допомогою фундус-камери.

2 ОСНОВНІ ПІДХОДИ ДО КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ

2.1 Постановка задачі класифікації зображень

Спочатку доцільно сформувати загальну постановку задачі класифікації.

Маємо множину класів $c = \{c_1, \dots, c_M\}$. Зокрема, при $M = 2$ маємо бінарну класифікацію.

Також є множина прикладів $X = \{X_r | r = \overline{1, N}\}$ обсягу N, кожен з яких відноситься рівно до одного класу, і при цьому кожен приклад $X_r, r = \overline{1, N}$ характеризується певною множиною ознак, які в загальному випадку можна записати у вигляді вектору:

$$X_r = (x_1^r \dots x_m^r)^T$$

де $m -$ кількість ознак,

x_i^r –значення і-тої ознаки r-го екземпляру.

Також маємо набір міток класу для кожного з прикладів:

$$Y = \{Y^r | r = \overline{1, N}\}$$

Треба знайти таку функцію $f(\cdot)$, яка би відносила кожний приклад з множини X до одного з класів множини c:

$$f(X_r) = \widehat{Y^r}$$

При цьому вона також повинна мінімізувати значення функції втрат, визначеної формулою (2.1):

$$J(Y,\hat{Y}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(Y^i,\hat{Y^i}) \tag{2.1}$$

Де

$Y -$ множина справжніх міток класу,

$\hat{Y} -$ множина прогнозованих значень,

$\mathcal{L}(\cdot,\cdot)$ −функція помилки на одному екземплярі,

$Y^i -$ справжня мітка класу для екземпляру i,

$\hat{Y^i} -$ прогнозоване значення для екземпляру i.

Наведену функцію помилки можна задати багатьма способами, але завжди повинні виконуватись дві умови:

- $\mathcal{L}(\cdot,\cdot) -$ невід'ємно значна функція;
- $\mathcal{L}(\cdot,\cdot)$ тим ближча до 0, чим ближчі між собою значення її аргументів, і при цьому вона по кожному аргументу визначена принаймні на множині класів c.

Однією з найпопулярніших функцій помилок для задачі бінарної класифікації є крос-ентропійна функція, визначена формулою (2.2):

$$\mathcal{L}(Y^i,\cdot\,\hat{Y^i}) = -Y^i\ln\hat{Y^i} - (1-Y^i)\ln(1-\hat{Y^i}) \tag{2.2}$$

Зокрема, у випадку багато-класової класифікації, вона узагальнюється до наступного вигляду у формулі (2.3):

$$\mathcal{L}(Y^i,\cdot\,\hat{Y^i}) = -\sum_{k=1}^{m} Y_{c_k}^i \ln\widehat{Y_{c_k}^i} \tag{2.3}$$

де $Y_{c_k}^i -$ індикатор належності i-того екземпляру до класу c_k,

$\widehat{Y_{c_k}^{i}}$ – імовірність належності і-того екземпляру до класу c_k, яку видає модель.

Її перевагою є те, що вона містить єдиний мінімум, який, до того ж, є глобальним.

Переходячи конкретно до класифікації зображень, формально, кожне зображення $X_r, r = \overline{1,N}$ можна представити наступним чином:

$$X_r = \left\{a_{ijk}^r | i = \overline{1,n}, j = \overline{1,m}, k = \overline{1,h}\right\}$$

де n – висота зображення,

m – ширина зображення,

h – кількість кольорових каналів зображення,

a_{ijk}^r –значення відповідного пікселя.

Найчастіше розглядаються випадок кольорового зображення ($h = 3$) та випадок чорно-білого зображення ($h = 1$). При цьому зазвичай ще вважається, що $n = m$, тобто що зображення, які розглядаються, є квадратними.

2.2 Базові нейронні мережі – персептрони

Переходячи безпосередньо до методів розв’язання задачі класифікації, доцільніше за все розглядати алгоритми нейронних мереж. Це є набір алгоритмів машинного навчання, які за своєю структурою є більш складними, ніж традиційні алгоритми машинного навчання, як то, наприклад, логістична регресія чи к-найближчих сусідів, і напрямлені на імітацію роботи людского мозику. Частіше за все розглядають глибокі нейронні мережі, які, зокрема, утворюють окремий підклас машинного навчання, який називається «глибоке

навчання». Про те, що саме собою являють глибокі нейронні мережі, буде розповідено трохи пізніше. Зараз же головне коротко прояснити, чому для класифікації зображень доцільніше за все використовувати саме моделі на базі нейронних мереж а не традиційні алгоритми машинного навчання. За великим рахунком, основних причин для цього дві.

По-перше, якщо розглядати доволі велику модель нейронної мережі, то часто для неї збільшення розміру навчальних даних суттєво довше буде покращувати результати роботи моделі, ніж для традиційних алгоритмів машинного навчання. Зауважимо насамперед, що це застосовно не тільки для задачі класифікації зображень, але і для інших задач машинного навчання.

По-друге, глибокі нейронні мережі мають багато різновидів шарів і моделей, які дозволяють краще ніж традиційні алгоритми працювати з неструктурованими даними, як то текстові дані, аудіо-записи, тощо, зокрема – зображення, для яких є згорткові нейронні мережі.

Але все ж таки, для кращого розуміння того, як працюють згорткові нейронні мережі, які як раз доцільно використовувати для задачі класифікації зображень, та які детально розглядатимуться у підрозділі 2.3, доцільно спочатку розглянути базові моделі нейронної мережі – персептрон Розенблата.

Розглянемо спочатку модель одношарового персептрону [7]. Схематично її можна описати рисунком 2.1.

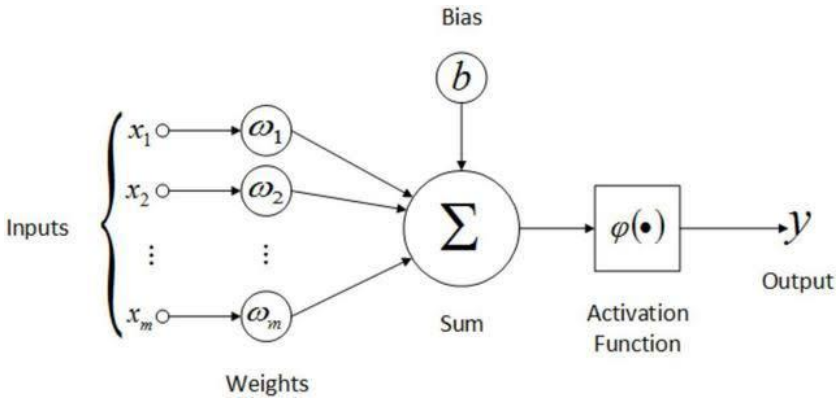


Рисунок 2.1 Одношаровий персептрон

Формально це можна описати наступним чином.

Маємо вектор ознак $\vec{x} = (x_1 \dots x_m)^T$, що характеризує даний об'єкт.

Треба спрогнозувати для нього мітку y. В залежності від задачі, це може буде мітка класу в задачі бінарної класифікації, числове значення в задачі регресії, тощо.

Для цього ми спочатку шукаємо лінійну комбінацію координат вектора $\vec{x}$, коефіцієнтами якої є координати певного вектора вагів $\vec{\omega} = (\omega_1 \dots \omega_m)^T$.

Потім, до отриманого значення ми додаємо коефіцієнт зсуву b.

І нарешті, на отримане значення ми діємо певною функцією активації $\varphi(\cdot)$, і отримуємо певну оцінку для нашої мітки, яку можна записати формулою (2.4):

$$\hat{y} = \varphi(\vec{\omega}^T \vec{x} + b) \tag{2.4}$$

Основні функції активації розглянемо трохи пізніше.

Зараз треба зазначити, що саме через свою відносну простоту, цей алгоритм доволі часто поступається іншим, більш складним моделям нейронних мереж, а інколи, навіть, традиційним алгоритмам машинного навчання. Що, в принципі, не дивно, оскільки більше того – в залежності від обраної функції активації, одношаровий персептрон може спокійно звестись до одного з традиційних алгоритмів машинного навчання, але про це ,знову ж таки, пізніше.

Вже суттєво частіше застосовуються так звані «глибокі нейронні мережі». Якщо коротко описувати їх суть, то вони крім вхідного та вихідного шарів ще містять певну кількість прихованих шарів, кожен з яких містить певну кількість так званих «нейронів». Найпростішим прикладом глибокої нейронної мережі слугує як раз багатошаровий персептрон [8]. Приклад багатошарового персептрону зображено на рисунку 2.2.

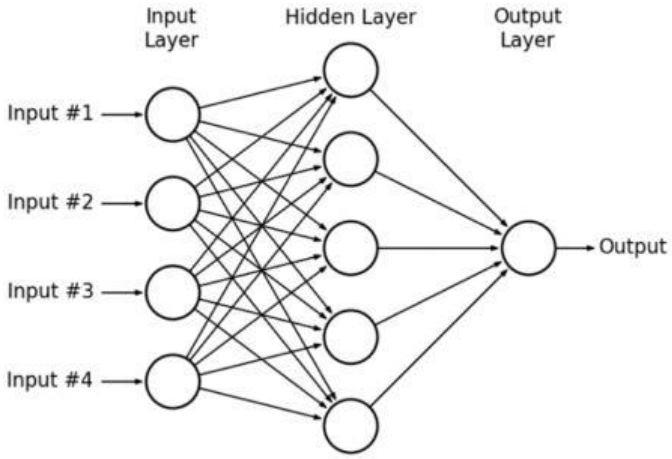


Рисунок 2.2 Багатошаровий персептрон

Загалом, принцип роботи багатошарового персептрону не сильно відрізняється від одношарового. Якщо у випадку одношарового персептрону з нашого прикладу в нас був тільки один нейрон, який, власне, належав вихідному шару, то у випадку багатошарового персептрону, ми просто спочатку повторюємо усі дії для кожного з нейронів першого прихованого шару, потім отримані значення на кожному з нейронів використовуємо як «новий вектор вхідних даних», і повторюємо цю процедуру на наступних шарах до тих пір, поки не дійдемо до останнього, вихідного шару, і не повторимо все те саме на ньому, отримавши остаточне вихідне значення або вектор значень(так, на вихідному шарі може бути кілька нейронів, але це вже більше стосується окремих, більш «екзотичних» задач). Також зауважимо, що кількість шарів в нейронні мережі завжди рахується як кількість прихованих шарів + 1, де ця одиниця відповідає вихідному шару.

Розглянемо тепер основні функції активації.

– Лінійна функція виражається формулою (2.5):

$$\varphi(z) = z \tag{2.5}$$

Дана функція є найпростішим варіантом функції активації, проте її не бажано застосовувати на прихованих шарах. Це пояснюється тим відомим

фактом з математичного аналізу і лінійної алгебри, що композиція лінійних функцій так само буде лінійною функцією. Отже, якщо застосовувати цю функцію активації на прихованих шарах, то ми просто «дарма» будемо використовувати приховані шари до першого шару з нелінійною функцією активації або вихідного шару, оскільки на останньому перед застосуванням функції активації ми все рівно отримаємо певну лінійну функцію від вхідних ознак, яку би ми могли отримати так само, якби одразу після вхідного поставили зазначений шар, але ми ще змусили нейронну мережу визначати велику кількість параметрів, що вимагає певного часу та обчислювальних ресурсів. Отже, цю функцію доцільно застосовувати хіба що на вихідному шарі, і те – лише для задач регресії, в яких цільова змінна може приймати будь-яке значення.

– Сігмоідальна функція виражається формулою (2.6):

$$\varphi(z) = \frac{1}{1+e^{-z}} \tag{2.6}$$

Головною особливістю цієї функції є те, що її значення завжди належать інтервалу $(0,1)$, через що її вихід можна інтерпретувати як імовірність належності до одного з двох класів, що є корисним і активно використовується у задачах бінарної класифікації, якщо поставити її на вихідний шар.

– Softmax виражається формулою (2.7):

$$\overrightarrow{\varphi(\vec{z})} = \begin{pmatrix} \frac{e^{z_1}}{\sum_{i=1}^{n} e^{z_i}} \\ \vdots \\ \frac{e^{z_n}}{\sum_{i=1}^{n} e^{z_i}} \end{pmatrix} \tag{2.7}$$

де $\vec{z} = (z_1 \quad \dots \quad z_n)^T$ – вектор аргументів.

Дана функція в нас вже є вектор-функцію, і за великим рахунком є узагальненням сігмоідальної функції на випадок задач багато-класової класифікації (не складно побачити, що всі координати даної вектор-функції невід'ємні, і в сумі дорівнюють одиниці, що дозволяє інтерпретувати їх як імовірності належності відповідним класам). Цю функцію доцільно використовувати на вихідному шарі в задачі багато-класової класифікації.

– Гіперболічний тангенс виражається формулою (2.8):

$$\varphi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.8}$$

В свою чергу, ця функція вже приймає значення з інтервалу $(-1,1)$, що буде корисним здебільшого для певного класу задач, де її можна поставити на вихідний шар, хоча інколи її також ставлять на приховані шари.

– Rectified Linear Unit (ReLU) функція виражається формулою (2.9):

$$\varphi(z) = \max(0, z) = \begin{cases} 0, z \leq 0 \\ z, z > 0 \end{cases} \tag{2.9}$$

Нескладно побачити, що ця функція приймає лише невід'ємні значення, тому її доцільно використовувати в задачах регресії з невід'ємними значеннями цільової змінної, як то, наприклад, прогнозування ціни «чогось». Також, зазначимо, що переважна більшість моделей нейронних мереж для класифікації зображень, що розглядатимуться надалі (і не тільки вони), використовують її як функцію активації на прихованих шарах.

– Leaky Rectified Linear Unit функція виражається формулою (2.10):

$$\varphi(z) = \begin{cases} \alpha z, z \leq 0 \\ z, z > 0 \end{cases} \tag{2.10}$$

де $\alpha -$ доволі мале додатне число (часто береться рівним 0.01).

Дана функція є модифікацією звичайної функції ReLU. Головною її відмінністю є те, що на відміну від базової функції вона при негативних значеннях аргументу приймає не нульові а малі від'ємні значення. Її зазвичай застосовують на прихованих шарах, для того, щоб запобігти проблеми «вмираючого ReLU», суть якої полягає в тому, що при застосуванні звичайної функції ReLU, якщо в нас усі значення перед застосуванням функції активації виходять від'ємними, то після застосування функції активації стають рівними 0, і так, зокрема, якщо функція знаходиться на вихідному шарі, градієнт буде рівним нулю, що «поставить крапку» на подальшому навчанні. В свою чергу, для даної функції, градієнт при від'ємних значеннях аргументу дорівнюватиме α, тому в більшій, чи меншій мірі, але навчання продовжуватиметься.

Бачимо, що в залежності від обраної функції активації, одношаровий персептрон може звестись до певних традиційних алгоритмів машинного навчання. Так, якщо розглядати модель з попереднього розділу, то взявши в якості функції активацію – сігмоідальну, ми фактично отримаємо алгоритм логістичної регресії, а взявши лінійну функцію активації – лінійну регресію.

Розглянувши складові одношарового персептрону, постає питання пошуку вектору ваг $\vec{\omega}$ та зсуву b.

Для цього, в нейронних мережах застосовуються метод прямого поширення похибки (англ. forward propagation method) та метод зворотного поширення похибки (англ. backpropagation method), які в поєднання як раз і утворюють увесь процес навчання моделі.

Власне виглядатиме процес навчання наступним чином:

- Спочатку випадковим чином ініціалізуються значення $\vec{\omega}^0$ та b^0.
- Далі виконується крок forward propagation – на нашому навчальному наборі даних ми прогнозуємо мітки за допомогою моделі з

ініціалізованими в минулому пункті параметрами, і на базі цих міток обраховуємо значення загальної функції помилки $J(Y, \hat{Y})$.

– Далі, ми застосовуємо крок backpropagation – застосовуємо чисельні методи оптимізації для модифікації параметрів, проходячи від
– Повторюємо всі пункти заново.

Щодо чисельних методів оптимізації, найбільш базовим для таких задач є метод градієнтного спуску. Процес модифікації параметрів за його допомогою можна написати наступним ітераційним процесом у формулі (2.11) та (2.12) для вектору ваг та коефіцієнту зсуву відповідно:

$$\vec{\omega}^{k+1} = \vec{\omega}^{k} - \alpha \frac{\partial J(Y,\hat{Y})}{\partial \vec{\omega}} \tag{2.11}$$

$$b^{k+1} = b^{k} - \alpha \frac{\partial J(Y,\hat{Y})}{\partial b} \tag{2.12}$$

де α −невід'ємна величина, що визначає розмір «кроку» методу,

$\frac{\partial J(Y,\hat{Y})}{\partial \vec{\omega}}$ − градієнт функції втрат за вектором ваг,

$\frac{\partial J(Y,\hat{Y})}{\partial b}$ −похідна функції втрат за параметром зсуву.

Проте розмір кроку треба підбирати ретельно, бо якщо його зробити надто великим, то під час оптимізації алгоритм завжди буде «проходити скрізь» оптимальних значень параметрів, і як варіант, ніколи не буде до них збігатись. В свою чергу, якщо робити значення кроку надто малим, то алгоритм до нього буде дуже довго збігатись.

Зауважимо також, що в базовому варіанті градієнтного спуску крок вважається сталим, а на практиці, в більшості випадків застосовуються його модифікації, за якими величина кроку змінюється динамічно – зменшується по мірі того, значення параметрів починають наближуватись до оптимальних. Однією з таких модифікацій, напевно – найпопулярнішою, є оптимізатор Adam [9], псевдокод алгоритму якого зображено на рисунку 2.3.

Input: θ - the initial parameters, $\beta_1, \beta_2 \in [0, 1)$ - the decay parameters, ϵ - step size, δ - a small constant for numerical stability, m - sample size, C - the cost function.
Output: θ - the optimal values of the parameters
/* Initialize momentum variables */
$s \leftarrow 0$
$r \leftarrow 0$

/* Start the iterations */
while *stopping criteria not met* **do**
  /* Sample a mini-batch */
  $\{(x_1^{(t)}, y_i^{(t)})\}_{i=1}^m \leftarrow$ sample m examples from the training data.
  /* Compute the gradient ($\widehat{y}_i$: prediction, y_i: ground truth) */
  $g \leftarrow \frac{1}{m} \nabla_\theta \sum_{i=1}^m C(\widehat{y}_i, y_i)$
  /* Adaptive momentum */
  $s \leftarrow \beta_1 s + (1 - \beta_1) g$
  $\widehat{s} \leftarrow \frac{1}{1-\beta_1^t} s$
  /* Adaptive learning rate */
  $r \leftarrow \beta_2 r + (1 - \beta_2)(g \odot g)$
  $\widehat{r} \leftarrow \frac{1}{1-\beta_2^t} r$
  /* Update the parameters */
  $\theta \leftarrow \theta - \epsilon \frac{\widehat{s}}{\sqrt{\widehat{r}} + \delta}$
end

return θ

Рисунок 2.3 Псевдокод алгоритму Adam

Можна також побачити, що в наведеному псевдокоді фігурує нове поняття “mini-batch” або міні-набір, яке є доволі важливим. Суть його полягає в тому, що ми замість того, щоб одразу навчати модель на цілому наборі даних, розбиваємо його на кілька малих наборів, і почергово виконуємо наведені кроки навчання моделі на кожному з них окремо. Мотивація для такого підходу полягає у тому, що якщо ми навчаємо модель на цілому наборі даних, то ми за певний час робимо лише один крок модифікації параметрів. В свою чергу, розбиттям на такі міні-набори ми за той самий час робимо кілька кроків модифікації параметрів, що значно прискорює навчання моделі, при цьому особливо не впливає на його якість.

2.3 Згорткові нейронні мережі

Розглянемо тепер дуже важливий клас нейронних мереж, який в більшості випадків і використовується в задачах класифікації зображень, і не тільки в них – згорткові нейронні мережі. Однією з головних, якщо не головною мотивацією для їх створення і подальшого розвитку став дуже суттєвий недолік звичайних персептронів в задачі класифікації зображень – для роботи навіть з доволі малими зображеннями потрібна дуже велика нейронна мережа. Дійсно, розглянемо цю ситуацію на прикладі дуже відомого набору даних із зображенням рукописних цифр – MNIST. Кожне зображення цього набору даних є чорно-білим і має розмір 28 на 28. Для того, щоб можна було використати цей набір даних на звичайному персептроні, треба в певному сенсі «розпрямити» кожне зображення цього набору, перетворивши його на вектор, що має 784 координати, де кожна координата відповідає значенню певного пікселя. Навіть якщо розглянути одношаровий персептрон, з функцією активації Softmax, ми отримуємо наступну кількість параметрів для навчання: 784*10+10=7850. Це дуже велике число, враховуючи, що ми маємо доволі малі зображення, і при цьому ще використовуємо доволі просту модель. В більшості задач і зображення будуть більше, оскільки більший розмір дозволяє зберегти кращу деталізацію, яка в свою чергу надає більше інформації про деталі, що важливо при класифікації, і моделі бажано брати побільше, щоб мати можливість знаходити більш складні закономірності. Згорткові нейронні мережі же дозволяють зменшити кількість навчальних параметрів, при цьому ще виокремлюючи на зображенні найважливіші деталі.

2.3.1 Основні поняття згорткових нейронних мереж

Головним поняттям, на якому базуються згорткові нейронні мережі – є поняття операції «згортки» [10]. Наочний опис суті даної операції можна побачити на рисунку 2.4.

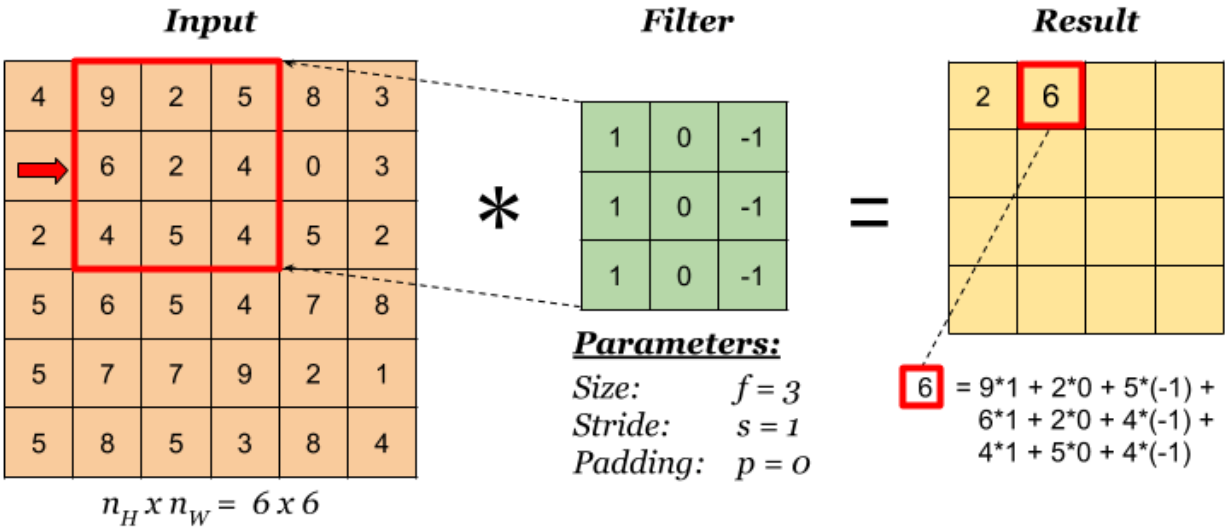


Рисунок 2.4 Приклад згортки зображення

Бачимо, що у нас виникає нова матриця. Зазвичай її називають ядро (англ. kernel) або фільтр (англ. filter). Її елементи є параметрами, які визначаються під час навчання нейронної мережі. Вона використовується для того, щоб виявляти певні ознаки на зображенні. Також можна побачити нові поняття «stride» та «padding», про які зараз власне піде мова.

Можна побачити, що в тій чи іншій мірі фільтр частіше застосовується до елементів, ближчих до центру зображення. Є один засіб, який дозволяє розв’язати цю проблему, і він називається – начинення (англ. padding) [11]. Суть цього методу в тому, що ми додаємо по краям пікселі фіксованого значення, і застосовуємо згортку вже до цього зображення. Зазвичай, значення пікселів по краях рівні нулю. Це називають нульовим начиненням (англ. zero padding). Приклад такого начинення зображено на рисунку 2.5.

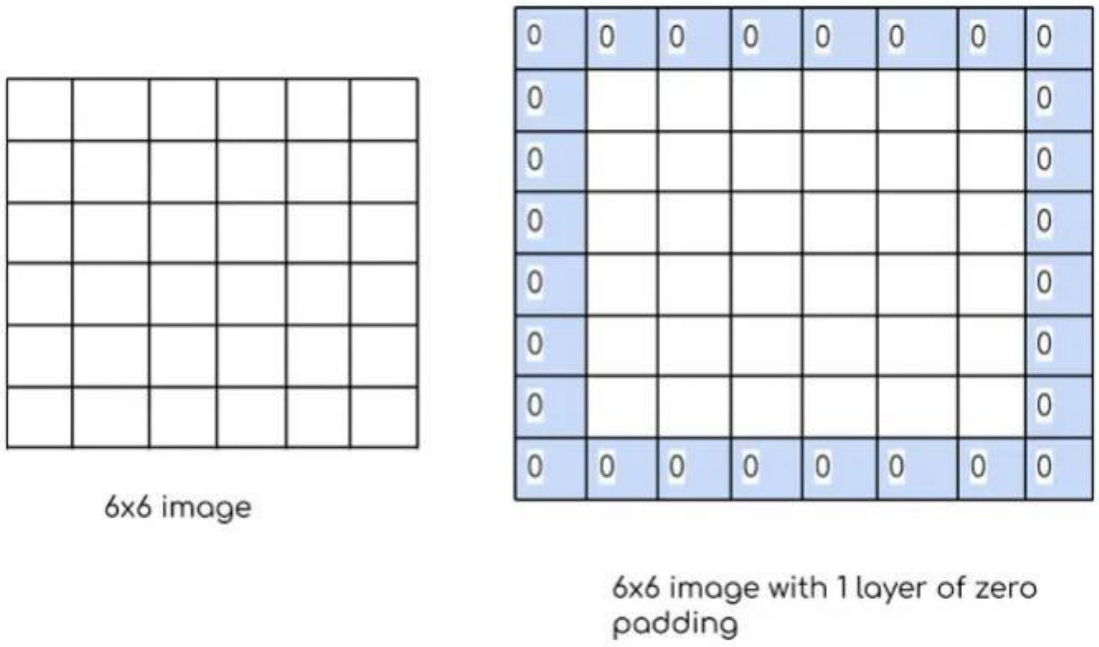


Рисунок 2.5 Приклад нульового начинення

Зауважимо, що розмір вихідного зображення зменшився. Інколи треба, щоб вихідне зображення було того ж розміру, як і вхідне. Зокрема, цього можна досягти за допомогою вибору правильного розміру начинення. Розглянемо цей випадок більш детально.

Нехай $n-$ розмір вхідного квадратного зображення, $f-$ розмір фільтру, який застосовується при згортці зображення, $p-$ розмір начинення.

З огляду на те, як відбувається процес згортки, можна побачити, що розмір вихідного зображення визначається формулою (2.13):

$$\hat{n} = n - f + 1 + p \tag{2.13}$$

Оскільки нам треба, щоб розмір вихідного зображення був рівний розміру вхідного, отримуємо наступне рівняння за p:

$$n = n - f + 1 + p$$

Отримуємо, що $p = f - 1$ – розмір начинення, при якому згортка не змінюватиме розмір зображення.

Також, говорячи про розмір вихідного зображення, треба згадати ще одне важливе поняття - крок згортки (англ. stride). Суть цього параметру $stide$ полягає в тому, що при згортці фільтр пересувається на $stride$ пікселів праворуч, та на $stride$ пікселів донизу [12]. Схематичний приклад згорток при різних значеннях параметра stride зображено на рисунку 2.6.

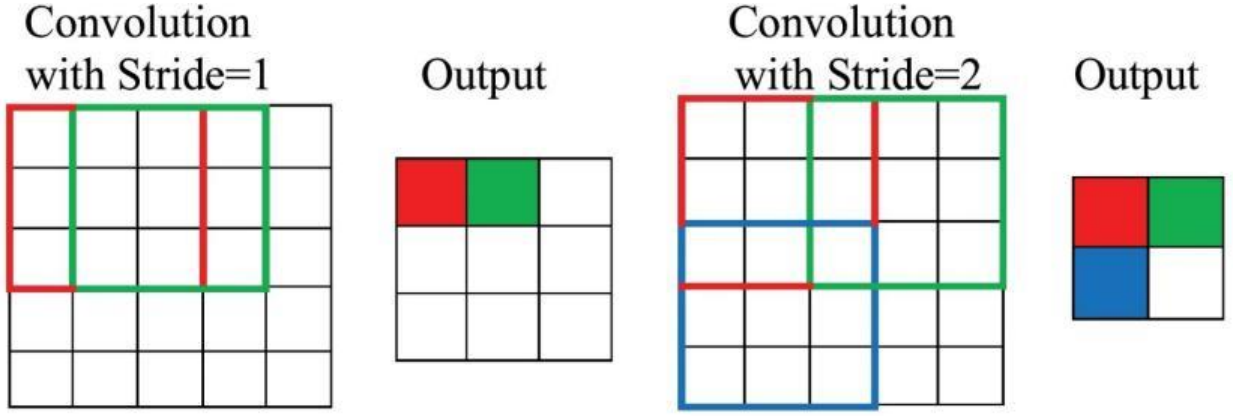


Рисунок 2.6 Згортка зі stride=1 та stride=2

В цьому випадку ми маємо, що розмір вихідного зображення визначатиметься формулою (2.14):

$$\hat{n} = \left[\frac{n-f+1+p}{s}\right] \tag{2.14}$$

Останнім доволі важливим поняттям, яке стосуються згорткових нейронних мереж, є шари стискання (англ. pooling layers). Суть їх в тому, що над вхідним зображенням проводиться процедура подібна до згортки, але із однією суттєвою відмінністю – замість того, щоб в якості вихідного значення для кожного пікселя брати лінійну функцію від значень пікселів, коефіцієнти якої є навчальними параметрами нейронної мережі, ми беремо певні функції без навчальних параметрів від значень тих пікселів. Найчастіше розглядають стискання максимумом (англ. max pooling), коли беруть максимальне зі

значень пікселів, та стискання середнім (англ. average pooling), коли беруть середнє-арифметичне значень пікселів [13]. Приклади таких стискань зображені на рисунку 2.7.

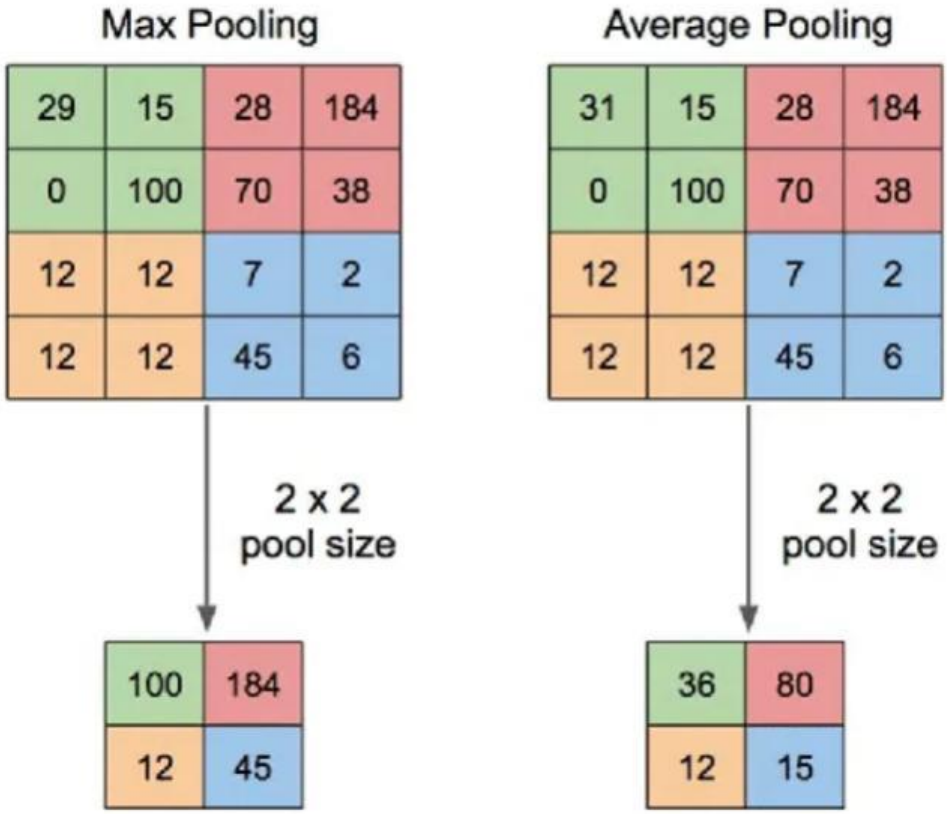


Рисунок 2.7 Приклад max та average pooling

Мотивація до застосування таких шарів полягає у тому, що, як зазначалось раніше, вони не вимагають навчання параметрів, тим самим економлять час та обчислювальні ресурси, і при цьому здатні зменшувати розмір зображення, при цьому ще виділяючи найбільш важливі ознаки. Так, наприклад, максимальне стискання вибирає найбільше значення пікселя серед наявних, яке на практиці дійсно може містити в собі найбільш важливу інформацію. Середнє стискання, в свою чергу, робить майже теж саме, тільки помірніше.

2.3.2 Приклади архітектур згорткових нейронних мереж

Розглянемо тепер приклади архітектур деяких нейронних мереж, що застосовувались до класифікації зображень, і доволі непогано себе там проявили [14].

Першою згортковою нейронною мережею, яку можна було би розглянути, є модель LeNet, яка була запропонована французьким науковцем Яном ЛеКуном у 1989 році. Проте розглядати будемо не базову версію даної мережі, а вже її подальшу модифікацію – LeNet5, запропоновану в 1998 році, яка використовувалась для розпізнавання рукописних цифр. Архітектуру даної мережі зображено на рисунку 2.8.

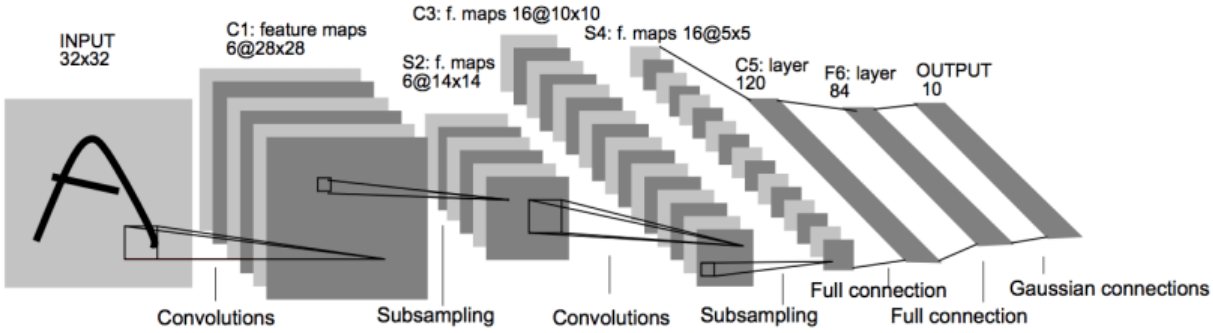


Рисунок 2.8 Архітектура моделі LeNet5

Наступною згортковою нейронною мережею, яку доцільно розглянути, є вже значно новіша і більша за своїми розмірами модель, яка в свою чергу призначалась для більш складної задачі, ніж розпізнавання рукописних цифр – AlexNet. Дана модель була розроблема Алексом Крижевським у співпраці з Іллєю Суцкевером, за допомогою якої Алекс переміг у змаганні ImageNet у 2012 році. Модель вже є доволі громіздкою, і розрахована на задачу класифікації з 1000 класів. Архітектуру цієї моделі зображено на рисунку 2.9.

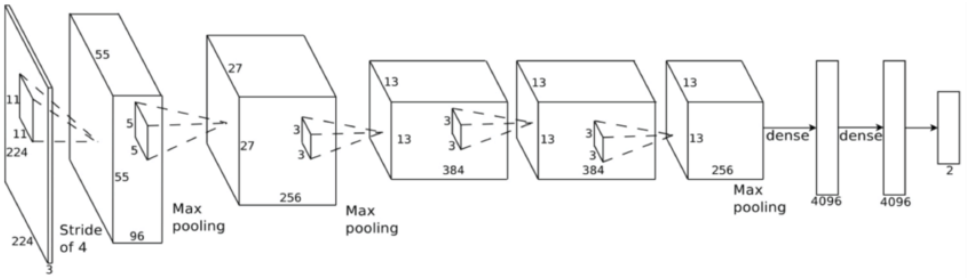


Рисунок 2.9 Архітектура моделі AlexNet

Розглянемо ще один приклад згорткової нейронної мережі, яка застосовувалась в задачі класифікації зображень – VGG. Насамперед зауважимо, що існують дві моделі VGG: VGG-16 та VGG-19. За своєю архітектурою вони є майже однакові, з єдиною відмінністю в тому, що модель VGG-19 є трохи більшою, тому, зазвичай, достатньо розглянути модель VGG-16. Вона була запропонована у 2014-му році Оксфордськими студентами Кареном Сімоняном та Ендрю Зіссерманом, і змогла заняти призові місця в змаганні ILSVRC у тому ж 2014-му році. Зокрема, вона призначалась для класифікації у випадку 1000 класів. Архітектуру моделі зображено на рисунку 2.10.

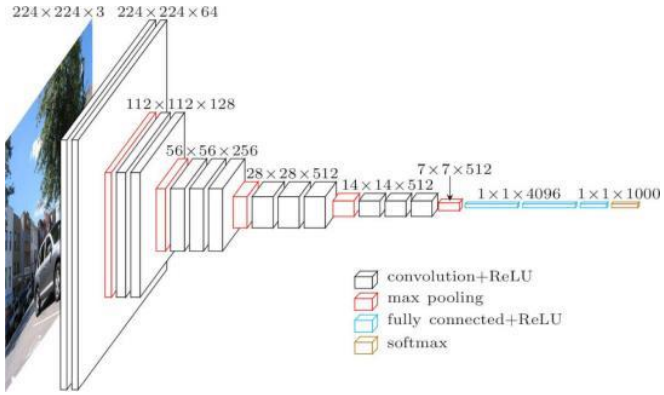


Рисунок 2.10 Архітектура моделі VGG-16

В усіх наведених архітектурах, можна побачити певну закономірність. Так, ми поступово зменшуємо розмір каналів зображення і збільшуємо їх кількість, при цьому чергуючи шари згортки та стискання, потім на певному етапі «розпрямляємо» зображення, записуючи усі ознаки, що лишились, у вигляді вектору, і надалі застосовуємо повно-з’єднані шари, як у звичайних персептронах.

Також варто ще зазначити один важливий тип нейронних мереж, а саме – залишкові нейронні мережі (англ. resnets). Суть даного класу нейронних мереж полягає у тому, що перед тим, як застосувати функцію активації до значень, ми додаємо до них значення з минулого шару (звісно, при цьому в них повинні бути однакові розміри). Зауважимо, що такий перехід називаються зрізом або сполученням швидкого доступу (англ. shortcut). Приклад такої роботи зображено на рисунку 2.11.

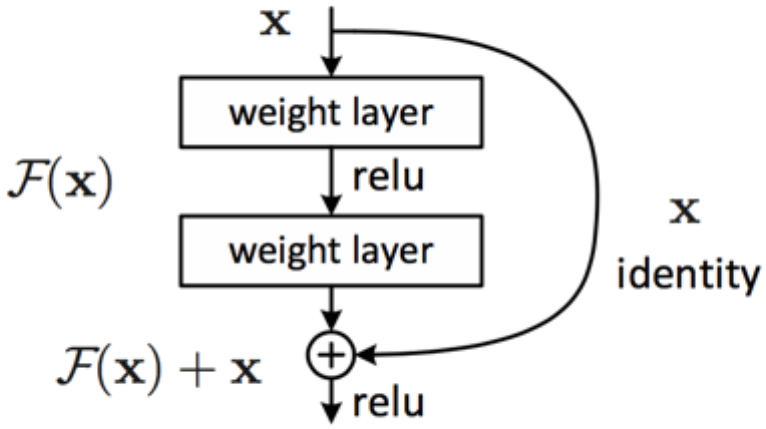


Рисунок 2.11 Сполучення швидкого доступу

В якості прикладу згорткової нейронної мережі, побудованої за таким принципом, можна навести на рисунку 2.12 архітектури моделей ResNet [15].

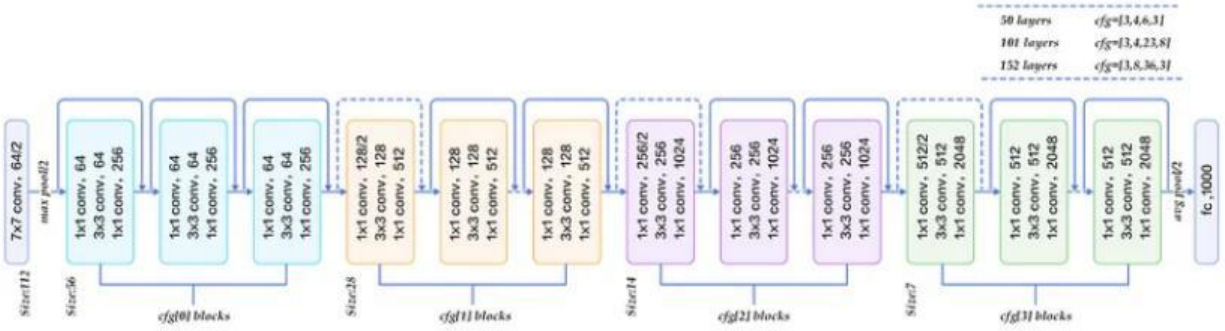


Рисунок 2.12 Архітектура моделей ResNet

2.4 Переносне навчання

В попередньому розділі ми, зокрема, розглянули декілька архітектур згорткових нейронних мереж, які свого часу вже були навчені, і добре себе показали при розв’язанні поставлених задач. Проте треба зазначити, що дані мережі були навчені на дуже великих наборах даних. На практиці же, далеко не завжди є в наявності дуже великий набір даних для навчання, а без нього , навчаючи модель «з нуля», скоріш за все ми не отримаємо від неї більш-менш хороших результатів роботи.

Виявляється, що на практиці буває доцільно взяти вже навчену модель для якоїсь іншої задачі, по типу тих, що ми розглядали в минулому розділі, зафіксувати значення її параметрів на всіх її шарах, окрім останнього або, можливо, кількох останніх, і навчати вже тільки їх. Власне в цьому і полягає сутність переносного навчання (англ. transfer learning). Практично, застосуванням такого підходу дійсно можна отримати модель, яка демонструє непогані результати.

Зауважимо при цьому, що застосування переносного навчання доцільне майже завжди, за виключенням тих випадків, коли в нас є власне дуже великий набір навчальних даних, про який ми згадували раніше. В нашому випадку це є по-особливому актуально і корисно, оскільки медичні дані, зокрема медичні знімки, не дуже часто знаходяться у відкритому доступі, на відміну від, наприклад, тих самих зображень тварин, а якщо і знаходяться – то у відносно невеликій кількості.

Насамкінець зазначимо, що чим більше в нас набір навчальних даних, тим більшу кількість шарів з кінця базової моделі доцільно брати для навчання.

2.5 Оцінка якості моделі

Після того, як модель вже навчена, постає питання того, а наскільки добре вона взагалі працює? Щоб відповісти на це питання, треба зокрема ввести певні показники якості моделі, що називаються метриками [16].

2.5.1 Основні метрики якості в задачах класифікації

Найпростішою метрикою якості в задачах класифікації є точність (англ. accuracy), і визначається вона наступним чином:

$$Accuracy = \frac{\text{кількість вірно класифікованих екземплярів}}{\text{загальна кількість екземплярів}}$$

Нескладно зрозуміти, що вона приймає значення на відрізку $[0,1]$, і чим більше її значення – тим краще. Зокрема, 0 означає, що модель абсолютно всі об’єкти класифікує неправильно, 1 – що абсолютно всі екземпляри класифікуються правильно.

Проте ця метрика має один суттєвий недолік – вона придатна до використання лише тоді, коли набір даних є збалансованим. Зазначимо, що збалансованість набору даних означає, що він містить однакову кількість об’єктів кожного з класів.

Якщо ж використовувати її у випадку, коли набір даних не є збалансованим то може виникнути ситуація, коли метрика буде мати доволі великі значення (наприклад, вище 90%), але модель насправді не буде нести в собі особливої або навіть жодної цінності щодо прогнозування.

В якості прикладу, припустимо, що ми розв’язуємо задачу бінарної класифікації, де треба визначити, є в людини певне захворювання, чи ні. При цьому, ми маємо набір даних, що містить 1000 екземплярів, 950 з яких відносяться до людей без цього захворювання, а 50 – до людей з ним. Вважатимемо, що особам без хвороби присвоюється мітка 0, а з хворобою – 1. Візьмемо в якості моделі ту, що завжди класифікує особу, як ту, що не має захворювання. Тоді ми отримаємо, що accuracy для даної моделі дорівнює 0.95, що є добрим результатом, але при цьому в моделі на практиці немає жодного сенсу, оскільки вона кожній людині просто присвоює одну і ту ж мітку класу.

Для цього вводяться нові метрики – інша точність (англ. precision) та повнота (англ. recall). Вони перш за все вводяться для задач бінарної класифікації (один клас має мітку 0, інший - 1), і визначаються вони формулами (2.15) та (2.16) відповідно:

$$Precision = \frac{TP}{TP+FP} \tag{2.15}$$

$$Recall = \frac{TP}{TP+FN} \tag{2.16}$$

де TP −True positive, кількість екземплярів, які насправді належать класу 1 і при цьому були класифіковані як екземпляр цього класу,

FP − False positive, кількість екземплярів, які були віднесені до класу 1, хоча насправді належать до класу 0,

FN − False negative, кількість екземплярів, які були віднесені до класу 0, але насправді належать класу 1,

TN −True negative, кількість екземплярів, що віднесені моделлю до класу 0, і при цьому насправді йому належать.

Зауважимо також, що за таких позначень, accuracy можна переписати формулою (2.17):

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad (2.17)$$

Якщо повернутись до нашого прикладу із захворюванням, то ми отримаємо, що для моделі, що класифікує кожного пацієнта як здорового, обидві метрики, precision та recall, дорівнюватимуть 0, оскільки $TP = 0$.

На значення якої метрики краще орієнтуватись – залежить від того, які помилки для нас є більш критичними. Якщо для нас більш критичними є значення FP , то для нас більш пріоритетними будуть великі значення $Precision$, якщо FN - $Recall$. Якщо ж нам бажано мінімізувати обидва види помилок, то треба тоді, щоб і Precision і Recall мали великі значення.

Для цього, зокрема, була запропонована метрика під назвою F1-міра наведена формулою (2.18):

$$F1 = \frac{2}{\frac{1}{Precision}+\frac{1}{Recall}} = \frac{2 \cdot Recall \cdot Precision}{Recall+Precision} \quad (2.18)$$

Бачимо, що F1-міра фактично є середнім гармонічним від $Precision$ і $Recall$. Такий вибір зумовлений відомою з математичного аналізу нерівністю середніх, згідно якої з 4-х видів середнього – арифметичного, геометричного, квадратичного та гармонічного, останнє в будь-якому випадку не перевищує значення інших, і при цьому рівність середніх досягається тоді і тільки тоді, коли усі числа рівні.

Отже, оскільки гармонічне середнє фактично є найменшим з усіх інших середніх, то для отримання доволі великих значень F1-міри, треба щоб і $Precision$ і $Recall$ мали доволі великі значення, не менші, ніж для отримання аналогічних значень інших середніх. На приклад, припустимо ситуацію, що для моделі ми отримали $Precision = 1$, $Recall = 0.5$. Якби ми брали їх

середнє арифметичне, то отримали би значення метрики 0.75, що може бути доволі прийнятним результатом в деяких випадках для метрики, що приймає значення з відрізку $[0,1]$, але якщо розглядати F1-міру, то ми би отримали її значення рівне $\frac{2}{3}$, що скоріш за все вже буде не дуже прийнятним результатом.

Також, доволі наочною метрикою якості моделі класифікації, є матриця неточностей (англ. confusion matrix). Така матриця в задачі бінарної класифікації має наступний вигляд:

$$\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix}$$

Як можна побачити на малюнку, у випадку бінарної класифікації елементами матриці є зазначені раніше числа TP,FP,TN,FN. В загальному випадку, це матриця, на головній діагоналі якої знаходяться кількості вірно класифікованих екземплярів для кожного з класів, а поза нею знаходяться кількості невірно класифікованих екземплярів, і при цьому сума значень матриці по кожному рядку дорівнює кількості екземплярів, які насправді належать відповідному класу, а по стовпцях – кількість екземплярів, які віднесені моделлю до відповідного класу. Ідеальним варіантом для нас був би той, при якому ця матриця виявляється діагональною. Також, у випадках з незбалансованим набором даних, доцільно використовувати так звану «нормалізовану матрицю неточностей». Це є матриця неточностей, в якій кожне значення ділиться на суму значень у відповідному рядку. Тоді кожне значення цієї матриці лежатиме на проміжку від 0 до 1, і сума значень в кожному рядку дорівнюватиме 1, незалежно від кількості екземплярів кожного класу у відповідному наборі даних, і в цьому випадку буде тим краще, чим ближче матриця буде до одиничної. Зокрема, у випадку бінарної класифікація дана матриця матиме наступний вигляд:

$$\begin{bmatrix} \frac{TN}{TN+FP} & \frac{FP}{TN+FP} \\ \frac{FN}{TP+FN} & \frac{TP}{TP+FN} \end{bmatrix}$$

2.5.2 Проблеми перенавчання та недонавчання

Також, говорячи про якість моделі машинного навчання, неможна не згадати дві доволі часті проблеми, що виникають під час навчання моделі – перенавчання (англ. overfitting або high variance), та недонавчання (англ. underfitting або high bias) [17].

Суть проблеми перенавчання полягає в тому, що модель навчається достатньо добре прогнозувати значення для даних, що були у навчальній вибірці, але при цьому показує дуже погані, або, принаймні, значно гірші результати при роботі з даними, які були відсутні у навчальній вибірці. Тобто модель не має нормальної узагальнюючої здатності.

Перенавчання моделі може бути зумовлено зокрема і найчастіше за усе наступними факторами:

- Малий розмір навчальної вибірки;
- Надто складна модель;
- Надто багато часу іде на навчання.

Для вирішення проблеми можна застосувати наступні заходи:

- Збільшити набір даних;
- Взяти більш просту модель;
- Зменшити час навчання;
- Застосувати до навчання регуляризацію.

Бачимо, що в нас виникає нове поняття – регуляризація, яке потребує певних роз’яснень [18]. Якщо описувати суть регуляризації коротко, то ми

замість того, щоб мінімізувати значення просто загальної функції втрат, мінімізуємо одночасно значення функції втрат і при цьому ще значення параметрів нейронної мережі. Продемонструємо це наочно на прикладі.

Спочатку розглянемо випадок одношарового персептрону. Як і раніше, в цьому випадку основними параметрами для навчання в нас є вектор вагів $\vec{\omega}$ та вектор зсуву b. Якщо раніше в нас пошук параметрів зводився до мінімізації виразу, наведеного формулою (2.19):

$$J(\vec{\omega}, b) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(y_i, \hat{y_i}) \tag{2.19}$$

то із регуляризацією ми мінімізуємо вираз, наведений формулою (2.20):

$$\hat{J}(\vec{\omega}, b) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(y_i, \hat{y_i}) + \frac{\lambda}{2m}\|\vec{\omega}\|_2^2 + \frac{\lambda}{2m}b^2 \tag{2.20}$$

де $\lambda > 0 -$ параметр, що визначає силу регуляризації,
$\|\vec{\omega}\|_2^2 = \sum_{i=1}^{m} \omega_i^2 - \ L_2$ - норма вектору $\vec{\omega}$.

Суть даного рішення полягає в тому, що якщо покласти $\lambda = 0$, то ми повернемось до початкової задачі мінімізації, і в решті-решт отримаємо ту ж саму перенавчену модель, а якщо взяти λ дуже великим, то будь-які ненульові значення параметрів все рівно робитимуть доволі значний внесок до цільової функції, через що для мінімізації виразу вони виходитимуть якщо і не нульовими, то дуже близькими до нуля. Якщо ж брати λ не нульовим, і не дуже великим, то під час розв’язання цієї задачі доведеться шукати компроміс між тим, щоб і втрати при прогнозуванні були відносно невеликі, і при цьому значення параметрів не становились дуже великими, що вже завадить нейронній мережі так добре «підлаштовуватись» під навчальні дані, і тим самим дозволить запобігти перенавчання. Також можна ще додати, що якщо

параметри будуть близькими(рівними) нулю, то сама модель стане простіше, а такій вже складніше перенавчатись.

Зауважимо також, що наведений вище вираз відповідає L_2- регуляризації, оскільки ми застосовуємо L_2- норму вектора $\vec{\omega}$. Також інколи застосовується L_1- регуляризація, яка полягає у мінімізації виразу, визначеного формулою (2.21):

$$\hat{J}(\vec{\omega}, b) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y_i, \hat{y}_i) + \frac{\lambda}{m}\|\vec{\omega}\|_1 + \frac{\lambda}{m}|b| \qquad (2.21)$$

де $\|\vec{\omega}\|_1 = \sum_{i=1}^{m}|\omega_i| - \ L_1$ - норма вектору $\vec{\omega}$.

На практиці спостерігається наступна різниця: якщо L_2- регуляризація робить значення незначних параметрів доволі малими по абсолютній величині, то L_1- регуляризація в свою чергу робить їх строго рівними нулю.

У випадку глибоких нейронних мереж все цілком аналогічно, тільки штраф може накладатись не тільки на тренувальні параметри останнього вихідного шару, але і на тренувальні параметри інших шарів.

Також, коли ми говоримо про позбавлення моделі від перенавчання, треба також згадати метод виключення (англ. dropout) для регуляризації. Якщо коротко пояснювати його суть, то ми під час навчання, на кожному кроці, випадковим чином виключаємо певні нейрони з нейронної мережі (під «виключанням» мається на увазі те, що незалежно від того, які значення цей нейрон отримує на свій вхід, його вихід дорівнює нулю). Приклад такого підходу можна зобразити на рисунку 2.13.

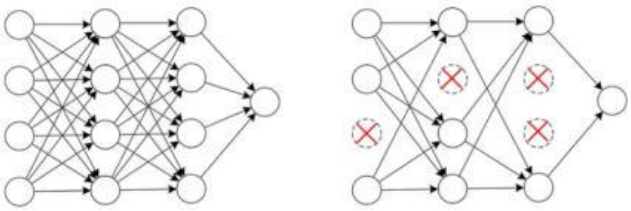


Рисунок 2.13 Приклад «виключення» нейронів

Якщо більш детально розглядати принцип його роботи, то ми для кожного шару вводимо імовірність того, що нейрон з нього треба відкинути, і на кожному кроці навчання, проходячись по кожному шару моделі, кожен його нейрон виключаємо із заданою імовірністю, і йдемо далі. При цьому, на кожному кроці можуть виключатись різні нейрони.

Інтиуїтивним аргументом для такого підходу є те, що виключення нейронів робитиме модель меншою за розміром на кожному кроці, яка в свою чергу має менше можливостей для «підлаштування» під навчальні дані, зокрема – для перенавчання. Іншим інтуїтивним аргументом є те, що оскільки кожен нейрон з заданого шару може бути з однаковою імовірністю виключений, нейронам з наступного шару, що прийматимуть його значення на свій вхід, не доцільно «робити значну ставку» на його значення, що матиме ефект у вигляді зменшення абсолютного значення коефіцієнта ваг при ньому, а це фактично є той ефект, що характерний для традиційної регуляризації.

Суть проблеми недонавчання в свою чергу полягає в тому, що модель не навчається добре прогнозувати значення навіть для екземплярів навчальної вибірки.

Недонавчання частіше за усе може бути зумовлене наступними факторами:

- Надто проста модель;
- Недостатньо часу іде на навчання моделі;
- Мала кількість ознак;
- Використовується надто сильна регуляризація.

Для вирішення цієї проблеми зазвичай використовуються наступні методи:

– Взяти більш складну та/або велику модель;
– Збільшити час навчання;
– Збільшити кількість ознак, якщо є можливість;
– Послабити регуляризацію.

Підводячи підсумки, суть проблеми недонавчання та перенавчання в задачі класифікації можна зобразити рисунком 2.14.

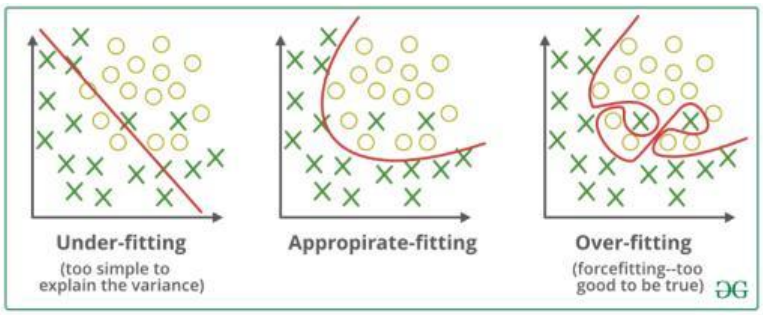


Рисунок 2.14 Перенавчання, недонавчання та оптимальне навчання

2.5.3 Збільшення даних для навчання

Говорячи про класифікацію зображень, ще необхідно згадати про таку річ як збільшення навчальних даних (англ. data augmentation).

Розповідаючи про переносне навчання, ми казали, що у випадку наявності лише маленького набору даних, можна брати за основу вже існуючі глибокі нейронні мережі, навчаючи при цьому лише останній або кілька останніх їх шарів, і отримувати доволі непогані результати. Проте, доволі часто набір навчальних даних можна в межах здорового глузду збільшити. Власне для цього і застосовується техніка, яка називається збільшенням даних.

Суть даної техніки полягає в тому, що ми можемо застосувати певний набір перетворень до даних, які з одного боку зроблять дані відмінними від вже наявних, а з іншого боку - залишать характерні ознаки, на базі яких приклади відносяться до певного класу.

Збільшення даних можна взагалі застосовувати не тільки до зображень, але саме на їх прикладі демонстрація роботи даної техніки є найбільш наочною. Так, у випадку зображень є принаймні наступні способи, за допомогою яких з одного зображення можна отримати декілька, при цьому зберігаючи в більшості випадків характерні ознаки:

- Горизонтальне/вертикальне віддзеркалювання зображення;
- Інші повороти зображення;
- Випадкові обрізання та/або масштабування зображення;
- Зміна яскравості, насиченості кольорів, контрастності, кольорової зображення;
- Додавання випадкового шуму та/або розмиття до зображення.

Зауважимо, що це далеко не всі засоби збільшення набору даних. Приклад такої техніки зображено на рисунку 2.15.

Рисунок 2.15 Приклад збільшення даних

Мотивацією для застосування даної техніки є те, на прикладі зображень, що якщо людина в більшості випадків одразу би помітила після таких перетворень, що це по суті трохи змінене, але одне й те саме зображення, то нейронна мережа покладається на значення пікселів зображення, а наведені перетворення дозволяють з одного боку – зберегти на зображенні основні ознаки, що відповідають за його належність до певного класу, а з іншого – змінити значення відповідних пікселів так, що нейронною мережею воно буде

сприйматись як кардинально нове зображення, з яким вона ніколи раніше не працювала.

Проте зауважимо, що не дивлячись на усі можливості, які перед нами відкриває дана техніка, вона має і свої недоліки. Зокрема, одним з них є той факт, що не завжди вихідне зображення після перетворення може бути якісним та придатним до подальшого застосування, оскільки воно може втрачати, наприклад, під час випадкового обрізання/масштабування характерні ознаки, що визначають приналежність його до певного класу.

Також, зауважимо, що розширення даних можна застосовувати не тільки для всього набору даних. Зокрема, цю техніку можна використовувати для вирішення проблеми незбалансованих класів у наборі даних [19], про яку частково розповідалось в розділі 2.5.1, застосовуючи дані процедури до меншого або менших з класів. Але в цьому випадку дану техніку треба застосовувати по-особливому з розумом та обережно, оскільки в деяких випадках вона може вносити небажані ознаки, які впливатимуть на роботу моделі. Так, якщо, наприклад, ми застосовуємо однакову зміну кольорової гами до зображень меншого з класів, і при цьому зображень у наведеній кольоровій гамі нема в інших класах, або вони є, проте їх кількість відносно незначна, то модель може навчитись непогано виявляти екземпляри цього меншого класу, але при цьому вона, можливо, буде зокрема спиратись на кольорову гаму зображення, чого не повинно бути.

Висновки до розділу 2

В даному розділі було розглянуто теоретичні відомості щодо нейронних мереж, як базових персептронів, так і більш придатних до класифікації зображень згорткових нейронних мереж: основні їх компоненти,

принципи роботи, наведено декілька прикладів архітектур згорткових нейронних мереж, які свого часу вже були навчені та непогано себе показали в задачі класифікації зображень. Також було розглянуто найбільш важливі аспекти побудови моделей для класифікації зображень: постановка задачі, основні метрики якості в задачі бінарної класифікації, а також засоби оцінки та покращення моделей, як то засоби розв’язання проблеми перенавчання та недонавчання, збільшення розміру набору навчальних даних, зокрема – для вирішення проблеми незбалансованого набору даних, що є доволі актуальним і важливим у випадку роботи з медичними зображеннями, а також розглянуто техніку переносного навчання, яку у випадку задачі класифікації зображень доцільно використовувати ледве не завжди за виключенням тих випадків, коли маємо дуже великий набір даних для навчання.

3 ВЛАСНА ПРОГРАМНА РЕАЛІЗАЦІЯ ЕКСПЕРИМЕНТУ

3.1 Обгрунтування вибору мови програмування, середовища написання коду та бібліотек

Перед тим, як перейти безпосередньо до проведення експериментів, треба обрати мову програмування та середу розробки.

Щодо вибору мови програмування, особливих сумнівів не було в том, що доцільніше за все брати мову програмування Python. Є багато аргументів за те, чому треба обирати саме її, але найбільш вагомими є два наступні: по-перше, мова є доволі простою, зі зручним синтаксисом, по-друге, в даній мові є велика кількість бібліотека та фреймворків, призначених для роботи з різнотипним даними, а також для побудови різних моделей машинного навчання, зокрема – нейронних мереж. Більш конкретно про ці бібліотеки, та про ті з них, що ми використовували – трохи згодом.

Другим важливим питанням в нашому випадку є вибір середовища написання коду для експерименту. Тут вибір автора пав на веб-середовище Google Colab. Перевагою цього середовища є по-перше те, що воно безкоштовно і загалом без попередніх завантажувань дозволяє редагувати блокноти Jupyter, які є дуже зручними як для написання коду, оскільки дозволяють запускати окремі його шматки незалежно від інших, що може суттєво економити час виконання, якщо треба щось підправити в одному місті, так і для роботи з візуалізацією даних, якої в нашому випадку теж буде немало. По-друге, цей сервіс дозволяє нам використовувати графічні процесори різної потужності, що, забігаючи наперед, буде в нашому випадку просто життєво-необхідним, оскільки дозволить значно прискорювати навчання наших моделей, і при цьому працювати з доволі великими наборами даних.

Говорячи про використані під час проведення експерименту бібіліотеки, доцільно зазначити наступні наступні з них:

- NumPy – бібліотека, що дозволяє працювати з даними, представленими у вигляді багатовимірних масивів. В нашому випадку використовувався частково для роботи з даними.
- Pandas - бібліотека що дозволяє працювати з табличним даними. Використовувалась для роботи з файлом, в якому містилась основна інформація про набір даних.
- OpenCV – бібліотека, призначена для цифрової обробки зображень. Безпосередньо з її використанням було реалізовано обробку зображень, призначених для навчання та перевірки роботи моделі, а також для збільшення набору навчальних даних також використовувалась вона.
- TensorFlow – в нашому випадку основна бібліотека для роботи з нейронними мережами та їх побудови.
- Matplotlib – бібліотека, призначена для візуалізації даних.
- os – модуль призначений для роботи з операційною системою. В нашому випадку він використовувався для зчитування зображень з директорії, в якій вони зберігались.
- Shutil – бібліотека, за допомогою якої зокрема можна переміщувати файли між директоріями.
- Sklearn – бібліотека, яка містить багато алгоритмів машинного навчання, а також просто корисних для машинного навчання алгоритмів, зокрема – обчислення метрик якості та деякі алгоритми для роботи з даними. В нашому випадку використовувалась для обчислення матриці неточностей, а також для «перемішування» даних в наборі.

– Seaborn – бібліотека, яка також використовується для візуалізації даних.

3.2 Аналіз та обробка даних

В якості базового набору даних для навчання було використано набір даних Ocular Disease Recognition, який знаходиться у відкритому доступі на платформі Kaggle. Цей набір містить 7000 зображень очного дна з лівого та правого ока. Значною перевагою цього набору даних є те, що в ньому містяться зокрема знімки очного дна, на яких одночасно було виявлено ознаки кількох захворювань. Це є добре, тому що дозволить нашій нейронній мережі отримати кращий «досвід» в тому сенсі, що вона навчиться відрізняти не тільки знімок із відповідним захворюванням від знімку здорового ока, але і знімок із захворюванням від знімку із відсутністю захворювання, в тому числі – коли немає конкретно заданого захворювання, але є інші, а не тільки знімки здорового ока.

На рисунку 3.1 зображено файл з даними, в якому міститься інформація про знімки.

	ID	Patient Age	Patient Sex	Left-Fundus	Right-Fundus	Left-Diagnostic Keywords	Right-Diagnostic Keywords	N	D	G	C	A	H	M	O
0	0	69	Female	0_left.jpg	0_right.jpg	cataract	normal fundus	0	0	0	1	0	0	0	0
1	1	57	Male	1_left.jpg	1_right.jpg	normal fundus	normal fundus	1	0	0	0	0	0	0	0
2	2	42	Male	2_left.jpg	2_right.jpg	laser spot, moderate non proliferative retinopathy	moderate non proliferative retinopathy	0	1	0	0	0	0	0	1
3	3	66	Male	3_left.jpg	3_right.jpg	normal fundus	branch retinal artery occlusion	0	0	0	0	0	0	0	1
4	4	53	Male	4_left.jpg	4_right.jpg	macular epiretinal membrane	mild nonproliferative retinopathy	0	1	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3495	4686	63	Male	4686_left.jpg	4686_right.jpg	severe nonproliferative retinopathy	proliferative diabetic retinopathy	0	1	0	0	0	0	0	0
3496	4688	42	Male	4688_left.jpg	4688_right.jpg	moderate non proliferative retinopathy	moderate non proliferative retinopathy	0	1	0	0	0	0	0	0
3497	4689	54	Male	4689_left.jpg	4689_right.jpg	mild nonproliferative retinopathy	normal fundus	0	1	0	0	0	0	0	0
3498	4690	57	Male	4690_left.jpg	4690_right.jpg	mild nonproliferative retinopathy	mild nonproliferative retinopathy	0	1	0	0	0	0	0	0
3499	4784	58	Male	4784_left.jpg	4784_right.jpg	hypertensive retinopathy, age-related macular d...	hypertensive retinopathy, age-related macular d...	0	0	0	0	1	1	0	0

3500 rows × 15 columns

Рисунок 3.1 Інформація про пацієнтів

Щоправда, можна також побачити один недолік в цьому файлі – розмітка щодо захворювань для пацієнтів на підставі знімків обох очей. Тобто,

як в першому рядку, якщо у людини на лівому оці є ознаки катаракти, а праве око – здорове, то пацієнт буде розмічений як той, що має катаракту. Це могло би бути непогано, як би нам треба було дістати інформацію про загальний стан пацієнта. Але нам треба отримати інформацію за конкретним знімком очного дна, а в цьому випадку ця розмітка непридатна. Для того, щоб нормально розмітити кожен знімок, ми будемо використовувати ключові слова, щодо діагнозів для кожного знімку – колонки “Left-Diagnostic Keywords” та “Right-Diagnostic Keywords” у відповідній таблиці. Так, наприклад, проаналізувавши більш детально набір даних, можна прийти до висновку, що діабетичні ретинопатії відповідають усі зображення, де в полі з діагнозом міститься слово «retinopathy» за виключенням «hypertensive retinopathy» , «myopia retinopathy» та «myopic retinopathy», зображенням з катарактою відповідають ті, в яких є слово «cataract» тощо.

Також, після чистки даних, за рахунок видалення нерелевантних зображень, оновлений файл матиме вигляд, зображений на рисунку 3.2.

	Image_name	N	D	G	C	A	H	M	O
0	0_left.jpg	0	0	0	1	0	0	0	0
1	0_right.jpg	1	0	0	0	0	0	0	0
2	1_left.jpg	1	0	0	0	0	0	0	0
3	1_right.jpg	1	0	0	0	0	0	0	0
4	2_left.jpg	0	1	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...
6915	4689_right.jpg	1	0	0	0	0	0	0	0
6916	4690_left.jpg	0	1	0	0	0	0	0	0
6917	4690_right.jpg	0	1	0	0	0	0	0	0
6918	4784_left.jpg	0	0	0	0	1	1	0	0
6919	4784_right.jpg	0	0	0	0	1	1	0	0

6920 rows × 9 columns

Рисунок 3.2 Представлення інформації в оновленому файлі

В отриманому файлі, як і в початковому файлі, останні вісім колонок відповідають діагнозам, наприклад «N» - «Normal», тобто ніяких захворювань нема, «D» - «Diabetes», тобто одна зі стадій діабетичної ретинопатії, «C» - «Cataract», тобто катаракта, і так далі, ще 4 захворювання, які ми не розглядаємо. При цьому, остання колонка «O» - «Others» відповідає наявності певних захворювань, які є суттєво більше рідкими, ніж інші та не входять до 6-ти інших наведених у таблиці.

Переходячи до того, що робити далі, для катаракти та діабетичної ретинопатії ми розглядатимемо дві задачі бінарної класифікації, де клас 1 означає наявність відповідного захворювання, а клас 0 – його відсутність, включаючи в себе як здорове око, так і наявність іншого захворювання, можливо – кількох. Зауважимо також, що в певному сенсі більш доцільним варіантом була би реалізація однієї нейронної мережі, яка би на вихідному шарі мала 8 нейронів, кожен з яких відповідав би певній з наведених у таблиці міток, і якому би відповідала сігмоідальна функція активації. Проте є декілька недоліків. По-перше, з того, що не увійшло до записки, можна зазначити, що було проведено певні експерименти в даному напрямку, і з’ясовано, що в нас все ж таки надто малий набір даних для настільки складних задач, і доповнити усі категорії достатньою кількістю зображень з відкритих джерел нема можливості. По-друге, у випадку деяких захворювань більш доцільним є попередня локалізація та відокремлення певної частини очного дна за зображенням, з подальшим виявленням вже за зображенням відокремленої частини. Як приклад, можна навести випадок глаукоми – підвищення тиску рідини в середині ока, для виявлення якої доцільніше спочатку локалізувати та відокремити на зображенні диск зорового нерву, і вже за ним класифікувати наявність чи відсутність глаукоми. Подібний підхід, зокрема, було реалізовано, наприклад, в [21]. Тому, в перспективі подальших досліджень такий підхід був би більш раціональний, і тому, власне, будемо розглядати задачі бінарної класифікації.

Але перед тим, як перейти до навчання, доцільно ще зробити невеличку обробку даних, а саме – трохи обрізати зображення, прибравши чорну частину по краях, яка не несе в собі жодної цінної інформації для класифікації, змінити розмір зображення до 256 на 256 пікселів, а також попрацювати над насиченістю зображення, що дозволить вирішити проблему різного освітлення на зображеннях. Результат такої обробки можна побачити на рисунках 3.3 та 3.4 відповідно.

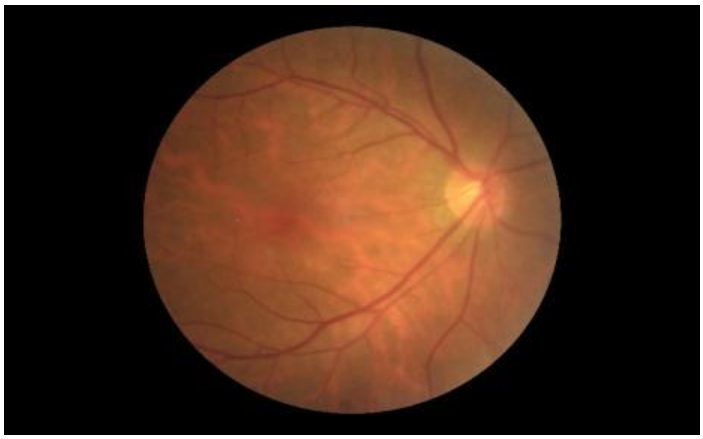


Рисунок 3.3 Зображення до обробки

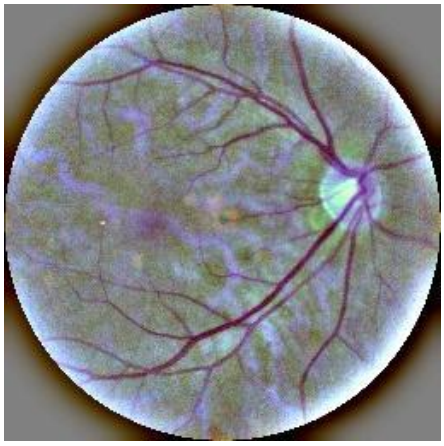


Рисунок 3.4 Зображення після обробки

Зауважимо, що такий підхід, щодо покращення насиченості зображення, пропонувався зокрема у [22].

Надалі постає проблема з тим, що для обох задач набори даних є доволі незбалансовані. Так, попередній розподіл класів для випадку катаракти та діабетичної ретинопатії можна побачити на рисунках 3.5 та 3.6 відповідно.

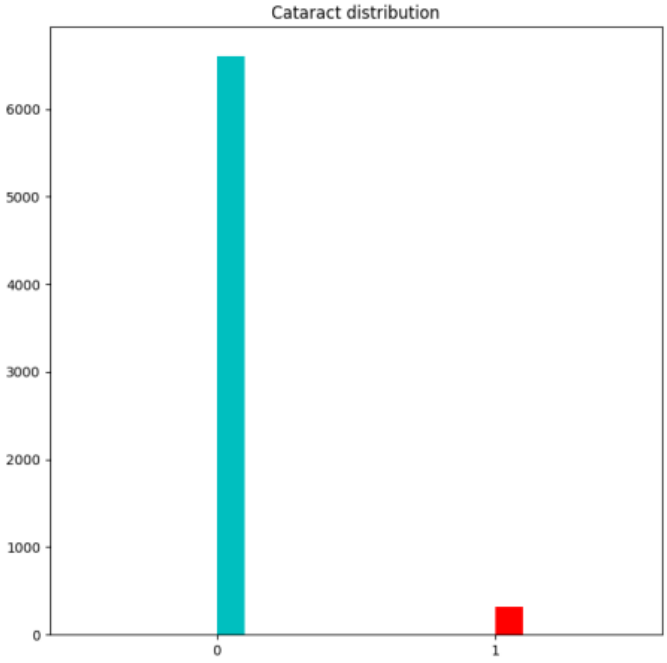


Рисунок 3.5 Розподіл класів у випадку катаракти

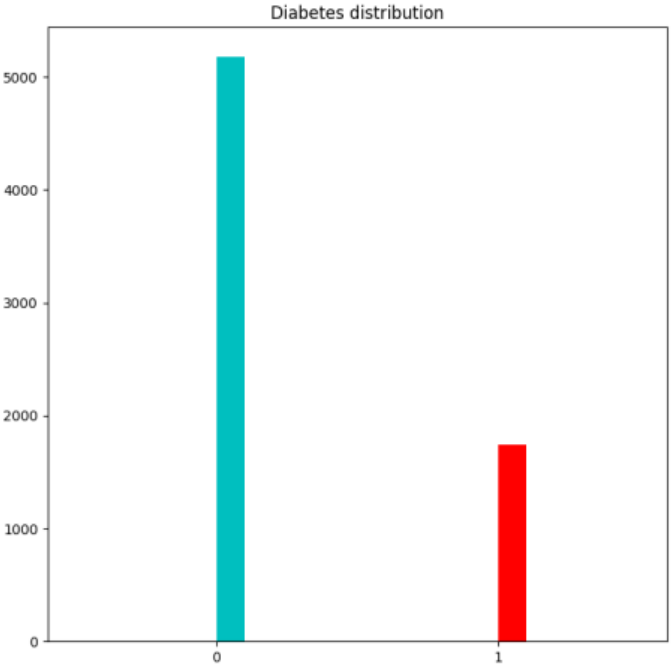


Рисунок 3.6 Розподіл класів у випадку діабетичної ретинопатії

Ця проблема вже в кожному з випадків буде вирішуватись по різному.

У випадку катаракти, забігаючи наперед, доволі хороші результати можна отримати навіть без вирішення проблеми незбалансованості набору даних, просто застосувавши аугментацію до всього навчального набору даних, за рахунок додавання випадкового гауссівського шуму, віддзеркалювання по

горизонталі та вертикалі, тим самим збільшивши навчальний набір даних в 4 рази. Приклади таких перетворень зображено на рисунку 3.7.

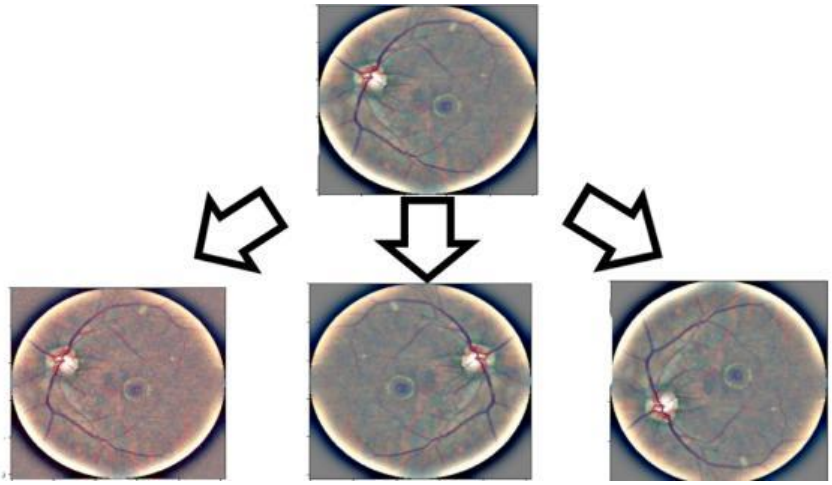


Рисунок 3.7 Перетворення для збільшення набору даних з катарактою

У випадку діабетичної ретинопатії все набагато складніше.

По-перше, для вирішення проблеми незбалансованості навчальних даних було прийнято рішення позичити навчальні данні, що відповідають діабетичній ретинопатії, з іншого набору даних, а саме – з набору даних зі змагання APTOS 2019 Blindness Detection на вже зазначеній раніше платформі Kaggle. Цей набір даних містив зображення різних стадій та здорових очей, проте перші ми об’єднали в один клас діабетична ретинопатія, оскільки, забігаючи наперед, в нашому випадку, коли треба навчитись виявляти певне захворювання у випадку, коли в якості альтернативи ставляться не лише знімки здорового ока, але й інші захворювання, доволі складно буде навіть у випадку бінарної класифікації навчити модель показувати більш-менш прийнятні результати, тому вже буде так. Зауважимо, що відповідні дані з нового набору даних було перевірено на наявність дублікатів зображень з базового набору даних, де все ж таки прийшли до висновку, що дублікатів немає, та ці дані можна використовувати. Також, зауважимо, що якщо до додавання даних з другого набору загальна кількість складала близько 7000 зображень, і відношення класів було приблизно 1 до 3, то після додавання отримали вже більше 10000 зображень, і відношення класів приблизно 1 до 2.

По-друге, певний тип аугментації, а саме – вертикальне віддзеркалювання, а також додавання гауссівського шуму до отриманого віддзеркаленого зображення, як горизонтального так і вертикального були застосовані лише до меншого з класів, а саме – до зображень з ознаками діабетичної ретинопатії. Приклад перетворень, що були застосовані до відповідного класу, наведено на рисунку 3.8.

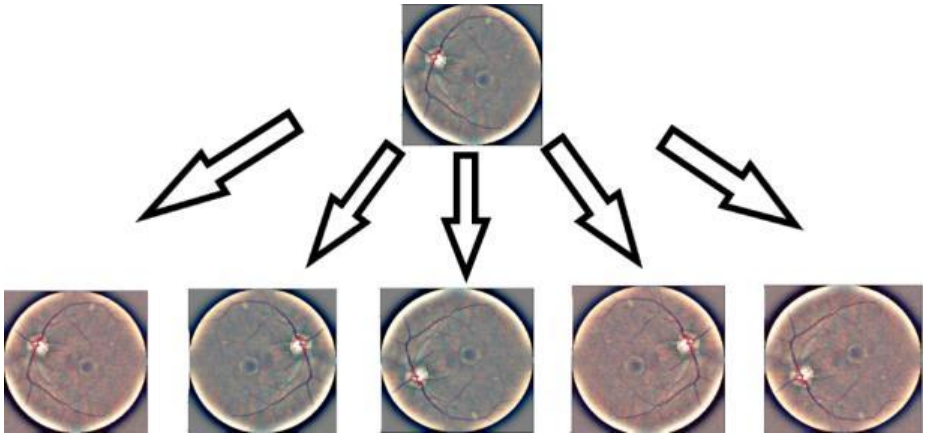


Рисунок 3.8 Попередня аугментація зображень з діабетичною ретинопатією

При цьому, до інших зображень було застосовано лише додавання гауссівського шуму та горизонтальне віддзеркалювання. Тим самим, ми фактично збільшили кількість зображень з діабетичною ретинопатією в 6 разів, без неї – в 3 рази. Після цього всього, для зображень з навчального набору даних, як базових – так і аугментованих, було двічі застосовано випадковий поворот на випадковий кут від 5 до 40 градусів в випадковому напрямку.

В решті-решт, ми отримали вже більш-менш збалансований набір даних, який, до того ж, значно більший за обсягом, і тому є кращим для навчання моделі.

3.3 Проведення експериментів для деяких моделей нейронних мереж та їх результати.

Для проведення експерименту, в якості базових моделей для переносного навчання розглядатимемо VGG-16, VGG-19 та ResNet50 без повно-з'єднаних шарів, а для останніх в обох випадках спробуємо підібрати свою архітектуру.

Спочатку розглянемо випадок з катарактою.

В якості архітектури моделей оберемо зображену на рисунку 3.9.

```
model=tf.keras.Sequential([base_model,
                           tf.keras.layers.Flatten(),
                           tf.keras.layers.Dropout(0.4),
                           tf.keras.layers.Dense(units=64,activation='leaky_relu'),
                           tf.keras.layers.Dense(units=4,activation='leaky_relu'),
                           tf.keras.layers.Dense(units=1,activation='sigmoid')])
```

Рис 3.9 Архітектура моделей для катаракти

Зауважимо, що в цьому випадку і надалі «base_model» відповідає як раз моделям наведених вище нейронних мереж, до яких буде застосовуватись переносне навчання.

Також, для навчання набір даних буде розбито на навчальну, валідаційну та тестову вибірки у співвідношеннях 80%, 10%, 10% відповідно.

Для моделі на базі VGG-16 отримаємо значення основних метрик, зображені у таблиці 3.1.

Таблиця 3.1 Результати моделі на базі VGG-16

	Train_value	Validation_value	Test_value
Accuracy	0.987	0.979	0.977
Precision	0.897	0.833	0.78
Recall	0.889	0.851	0.886
F1-score	0.893	0.842	0.83

Матриця неточностей, та її нормалізований варіант для тестової вибірки зображені на рисунках 3.10 та 3.11 відповідно.

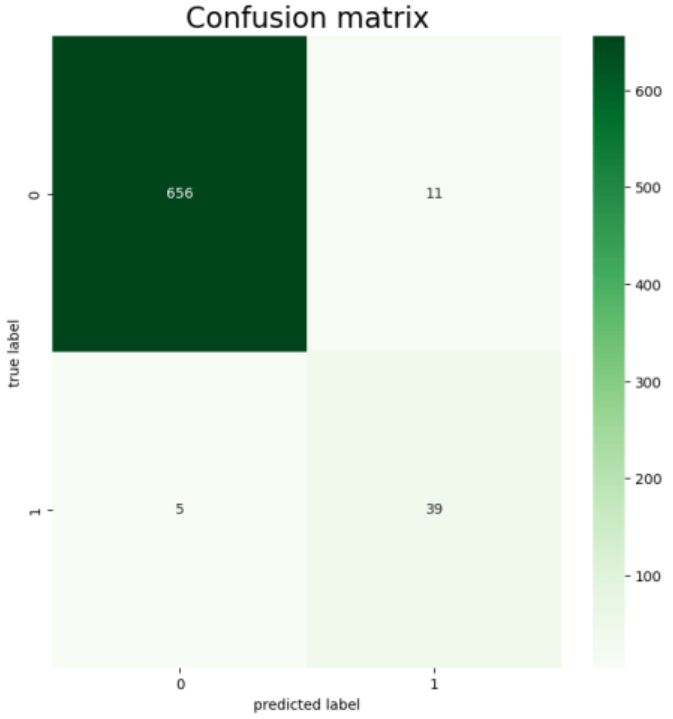


Рисунок 3.10 Матриця неточностей для VGG-16

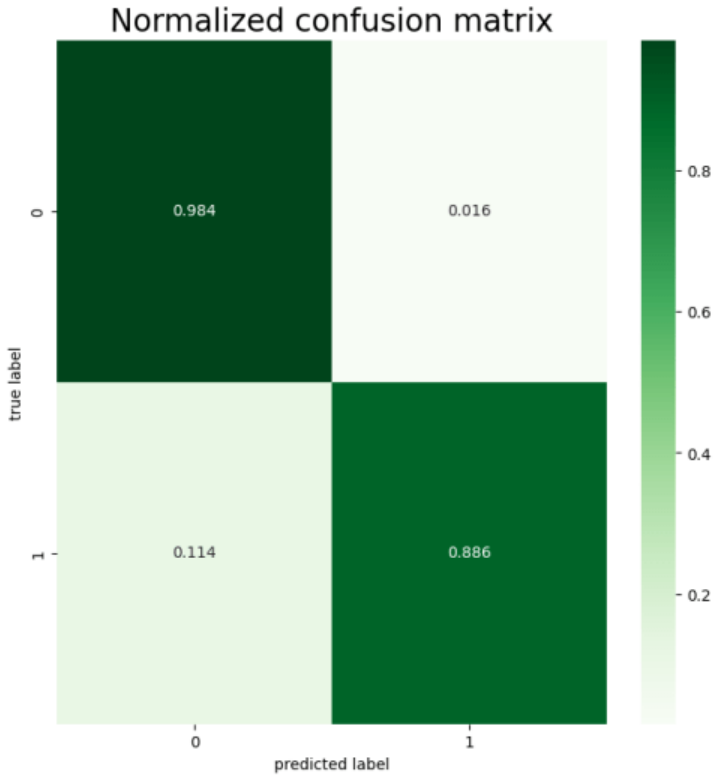


Рисунок 3.11 Нормалізована матриця неточностей для VGG-16

Для моделі на базі VGG-19 отримаємо значення основних метрик, зображені у таблиці 3.2.

Таблиця 3.2 Результати моделі на базі VGG-19

	Train_value	Validation_value	Test_value
Accuracy	0.991	0.979	0.977
Precision	0.926	0.921	0.868
Recall	0.925	0.745	0.75
F1-score	0.925	0.824	0.805

Матриця неточностей, та її нормалізований варіант для тестової вибірки зображені на рисунках 3.12 та 3.13 відповідно.

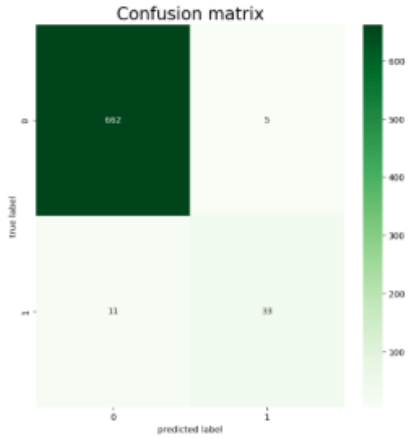


Рисунок 3.12 Матриця неточностей для VGG-19

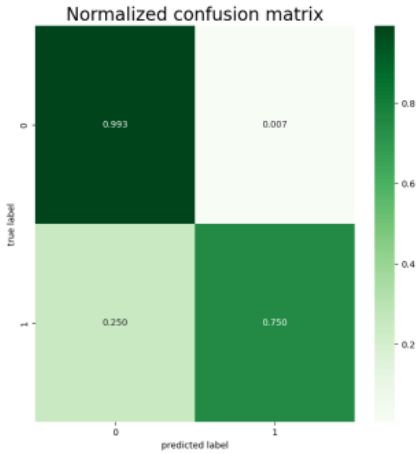


Рисунок 3.13 Нормалізована матриця неточностей для VGG-19

Для моделі на базі ResNet50 отримаємо значення основних метрик, зображені у таблиці 3.3.

Таблиця 3.3 Результати моделі на базі ResNet50

	Train_value	Validation_value	Test_value
Accuracy	0.985	0.969	0.975
Precision	0.882	0.705	0.861
Recall	0.874	0.915	0.704
F1-score	0.877	0.796	0.774

Матриця неточностей, та її нормалізований варіант для тестової вибірки зображені на рисунках 3.14 та 3.15 відповідно.

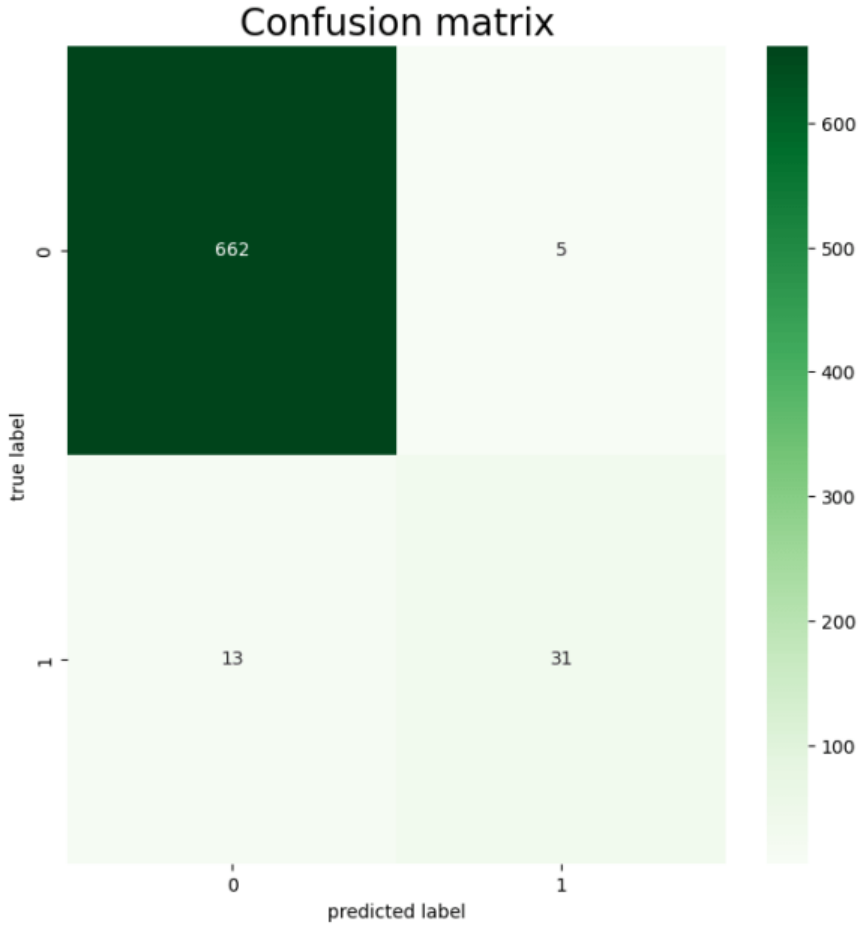


Рисунок 3.14 Матриця неточностей для ResNet50

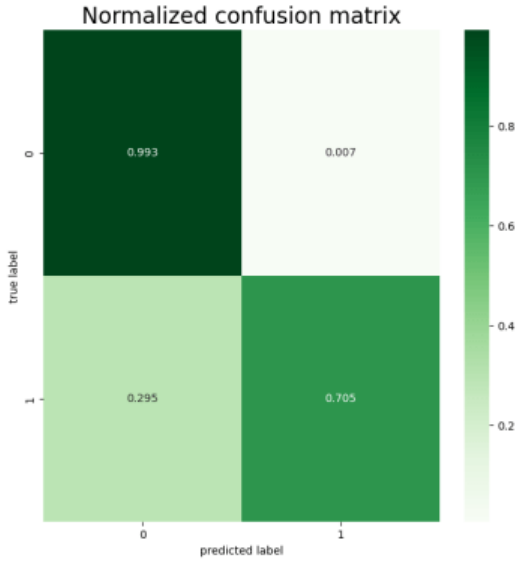


Рисунок 3.15 Нормалізована матриця неточностей для ResNet50

Тепер розглянемо випадок діабетичної ретинопатії.

В якості архітектури моделей оберемо зображену на рисунку 3.16.

```
model=tf.keras.Sequential([base_model,
                           tf.keras.layers.Flatten(),
                           tf.keras.layers.Dropout(0.5),
                           tf.keras.layers.Dense(units=256,activation='leaky_relu'),
                           tf.keras.layers.Dropout(0.25),
                           tf.keras.layers.Dense(units=32,activation='leaky_relu'),
                           tf.keras.layers.Dense(units=8,activation='leaky_relu'),
                           tf.keras.layers.Dense(units=1,activation='sigmoid')])
```

Рисунок 3.16 Архітектура моделей для діабетичної ретинопатії

Як і раніше, для навчання набір даних буде розбито на навчальну, валидаційну та тестову вибірки у співвідношеннях 80%,10%,10% відповідно.

Для моделі на базі VGG-16 отримаємо значення основних метрік, зображені у таблиці 3.4.

Таблиця 3.4 Результати моделі на базі VGG-16

	Train_value	Validation_value	Test_value
Accuracy	0.753	0.824	0.759
Precision	0.77	0.751	0.608
Recall	0.732	0.714	0.831
F1-score	0.751	0.732	0.702

Матриця неточностей та її нормалізований варіант зображено на рисунках 3.17 та 3.18 відповідно.

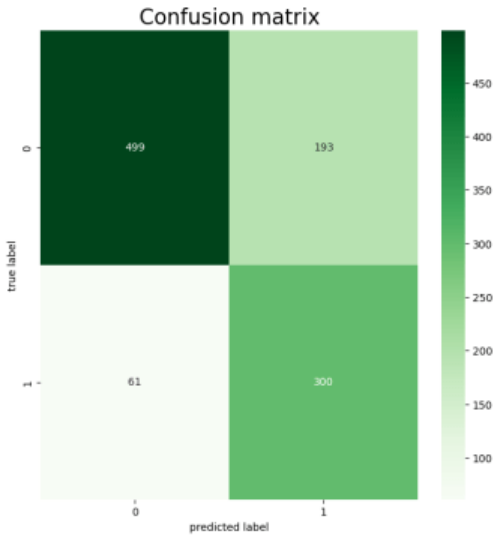


Рисунок 3.17 Матриця неточностей для VGG-16

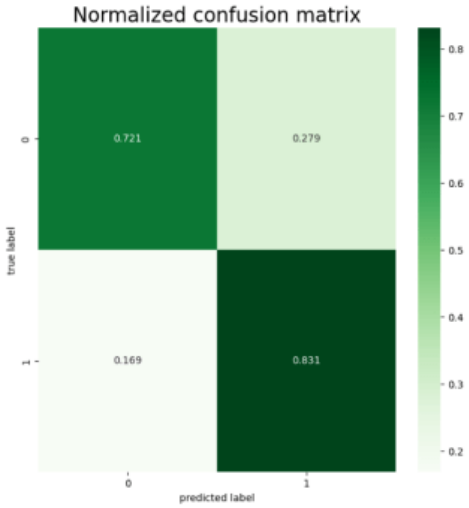


Рисунок 3.18 Нормалізована матриця неточностей для VGG-16

Для моделі на базі VGG-19 отримаємо значення основних метрик, зображені у таблиці 3.5.

Таблиця 3.5 Результати моделі на базі VGG-19

	Train_value	Validation_value	Test_value
Accuracy	0.755	0.801	0.801
Precision	0.779	0.68	0.687
Recall	0.723	0.771	0.773
F1-score	0.75	0.723	0.727

Матриця неточностей, та її нормалізований варіант для тестової вибірки зображені на рисунках 3.19 та 3.20 відповідно.

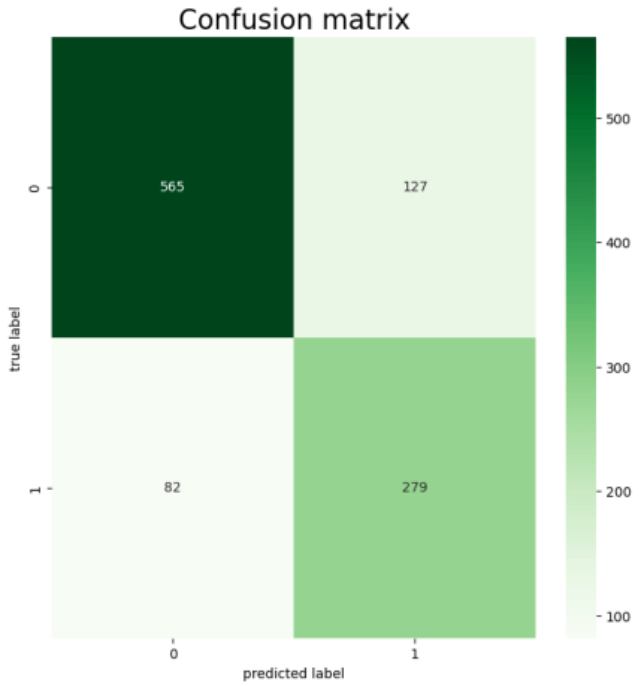


Рисунок 3.19 Матриця неточностей для VGG-19

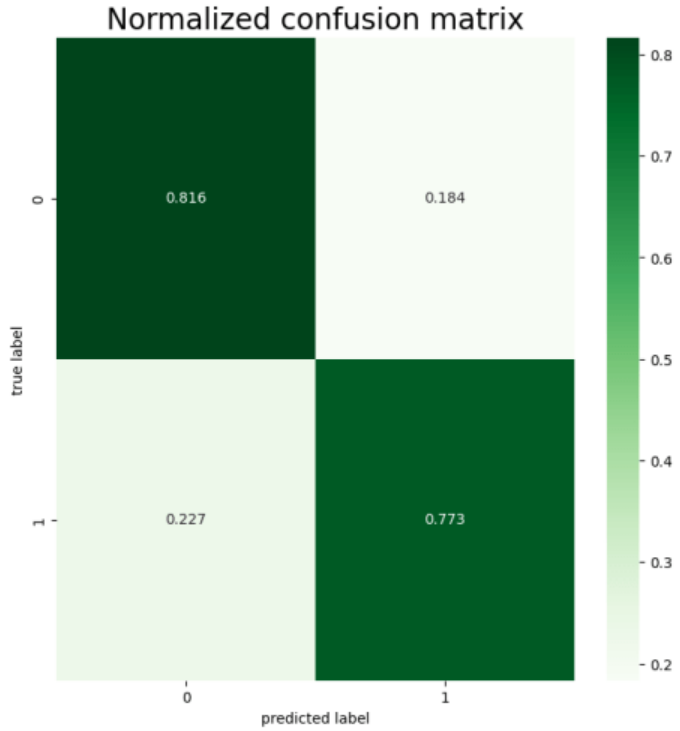


Рисунок 3.20 Нормалізована матриця неточностей для VGG-19

Для моделі на базі ResNet50 отримаємо значення основних метрик, зображені у таблиці 3.6.

Таблиця 3.6 Результати моделі на базі ResNet50

	Train_value	Validation_value	Test_value
Accuracy	0.782	0.743	0.763
Precision	0.811	0.579	0.611
Recall	0.743	0.86	0.853
F1-score	0.775	0.692	0.712

Матриця неточностей, та її нормалізований варіант для тестової вибірки зображені на рисунках 3.21 та 3.22 відповідно.

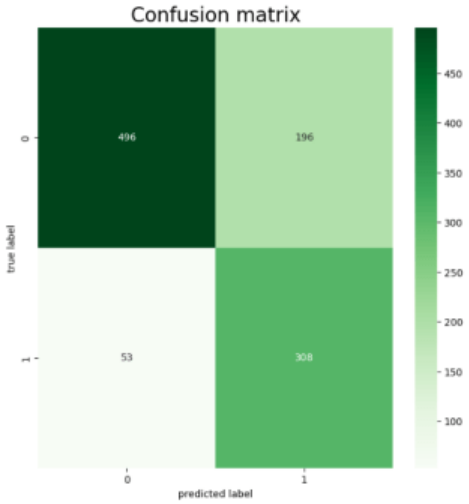


Рисунок 3.21 Матриця неточностей для ResNet50

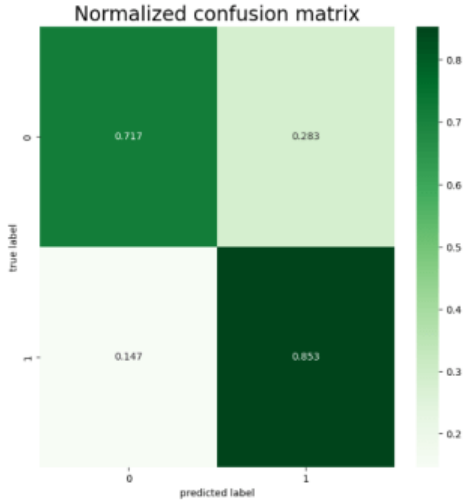


Рисунок 3.22 Нормалізована матриця неточностей для ResNet50

Підсумовуючи, можна навести таблицю 3.7, в якій наведено результати найкращих моделей, з урахуванням як безпосередньо значень метрик на тестовій вибірці, так і наявності або відсутності перенавчання чи недонавчання моделі.

Таблиця 3.7 Найкращі результати для обох випадків

	Випадок катаракти	Випадок діабетичної ретинопатії
Accuracy	0.977	0.801
Precision	0.78	0.687
Recall	0.886	0.773
F1-score	0.83	0.727

Висновки до розділу 3

В даному розділі було проведено попередню обробку даних, та на їх основі вже було побудовано декілька моделей нейронних мереж на базі існуючих за допомогою переносного навчання. Для того, щоб обрати кращі з них, було порівняно як значення основних метрик між собою. В результаті, можна сказати, що найкращою з моделей для виявлення катаракти є та, що побудована на базі моделі VGG-16, а для виявлення діабетичної ретинопатії – на базі VGG-19. При цьому, у випадку розпізнавання катаракти результати були значно кращі, ніж у випадку розпізнавання діабетичної ретинопатії, зокрема через те, що перша має суттєво більш помітні ознаки, які виокремлюють її значно більше, ніж будь що інше. Також, було зроблено висновки щодо перенавчання та недонавчання моделей, згідно яких моделі мають певне перенавчання, оскільки все ж таки значення метрик на навчальній

вибірці є кращими, ніж на валідаційній чи тестовій, але вони не настільки краще, щоб це було хоч якоюсь проблемою.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПП

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на виявленні захворювань очей у людей за зображенням очного дна.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування.

F_2 – вибір середовища розробки.

F_3 – вибір бібліотеки для побудови нейронних мереж.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python.

б) C++.

Функція F_2.

а) Google Colab.

б) Visual Studio.

Функція F_3:

а) TensorFlow.

б) PyTorch.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

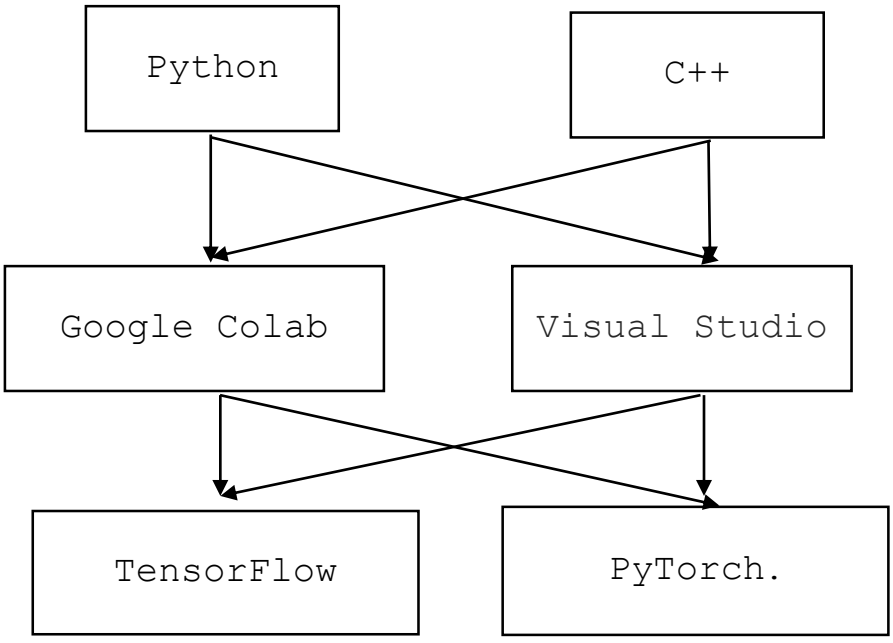


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантиів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Велика кількість бібліотек для роботи з даними та машинного навчання.	Код виконується повільніше.
	Б	Швидке виконання коду.	Відсутність такої великої кількості потрібних бібліотек.
F_2	*А*	Надійно працює зі складними проектами	Не підтримується багатьма мовами
	Б	Надійність	Додатковий час на інсталяцію та вивчення
F_3	*А*	Зручність відображення результатів виконання окремих блоків коду. Можливість використовувати хмарні графічні процесори та підвищений розмір оперативної пам’яті.	Відсутня можливість роботи без інтернету
	Б	Багато інструментів. Безпечна.	Підтримує одночасно лише одну мову програмування

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Перевагу даємо мові Python. Швидкість виконання коду в даному випадку не є настільки важливою, тому варіант Б відкидаємо.

Функція F_2:

Можливість застосовувати потужні графічні процесори та підвищену кількість оперативної пам’яті є доволі суттєвим, тому тут обираємо варіант А.

Функція F_3:

Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

$$F_1\text{а} - F_2\text{а} - F_3\text{а}$$
$$F_1\text{а} - F_2\text{а} - F_3\text{б}$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – швидкодія мови програмування;
- *X2* – об’єм пам’яті для обчислень та збереження даних;
- *X3* – час навчання даних;
- *X4* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	10000	14000	19000
Об’єм пам’яті	X2	Мб	420	128	64
Час попередньої обробки даних	X3	мс	4	3	2
Потенційний об’єм програмного коду	X4	кількість рядків коду	4000	2500	1000

За даними таблиці 4.3 будуються графічні характеристики параметрів – рисунок 4.2 – рисунок 4.5.

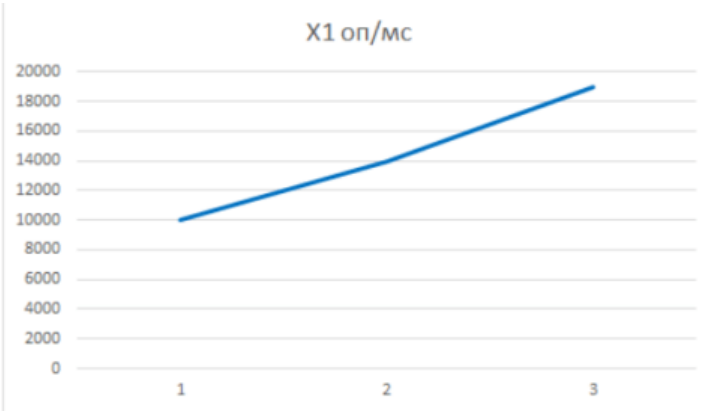


Рисунок 4.2 – X1, швидкодія мови програмування

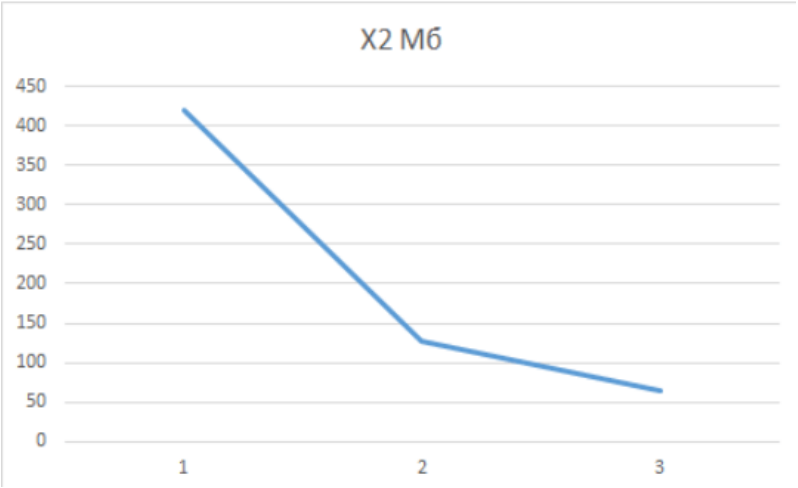


Рисунок 4.3 – X2, об’єм пам’яті

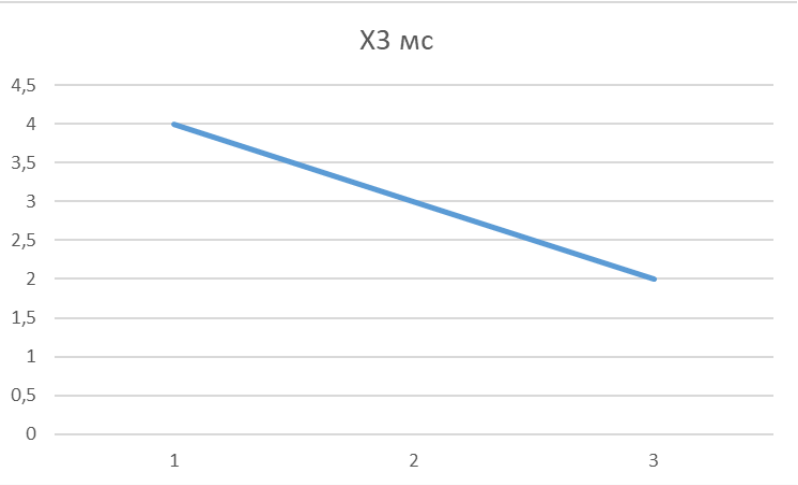


Рисунок 4.4 – X3, час попередньої обробки даних

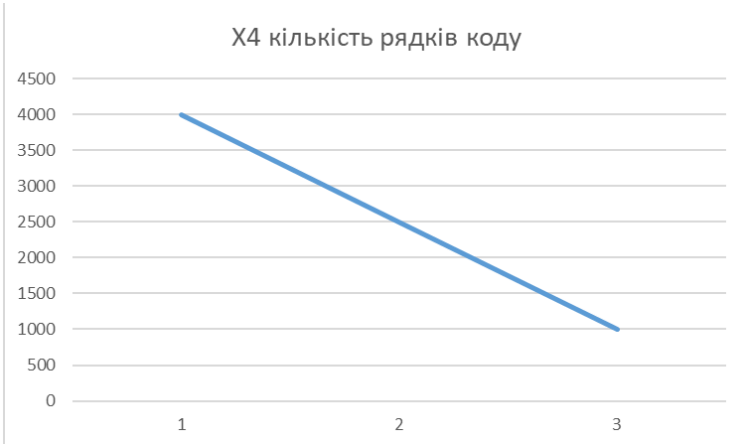


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці вимірy	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	2	1	1	2	2	2	1	11	-6,5	42,25
X2	Об’єм пам’яті	Mb	4	4	3	3	3	4	4	25	7,5	56,25
X3	Час попередньої обробки даних	мс	1	2	2	1	1	1	2	10	-7,5	56,25
X4	Потенційний об’єм програмного коду	Кількість рядків коду	3	3	4	4	4	3	3	24	6,5	42,25
	Разом		10	10	10	10	10	10	10	70	0	197

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \qquad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \tag{4.2}$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \tag{4.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197. \tag{4.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 197}{7^2(4^3 - 4)} = 0{,}754 > W_k = 0{,}67. \tag{4.5}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	<	>	<	<	<	<	<	0,5
X1 і X3	>	>	<	>	>	<	<	>	1,5
X1 і X4	>	>	<	<	<	<	<	<	0,5
X2 і X3	>	>	>	>	>	<	>	>	1,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	>	>	<	<	<	>	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \quad (4.9)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \quad (4.10)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{вi}}$	b_i^1	$K_{\text{вi}}^1$	b_i^2	$K_{\text{вi}}^2$
X1	1	0,5	1,5	0,5	3,5	0,21	12,25	0,21	44,875	0,21
X2	1,5	1	1,5	0,5	4,5	0,27	16,25	0,27	59,125	0,28
X3	0,5	0,5	1	0,5	2,5	0,16	9,25	0,16	34,875	0,16
X4	1,5	1,5	1,5	1	5,5	0,36	21,25	0,36	77,875	0,35
Всього:					16	1	59	1	213	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об’єм пам’яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об’єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	10000	3	0,21	0,63
F2	А	X2	64	6	0,28	1,68
F3	А	X3	128	9	0,16	1,44
	Б	X4	1000	4	0,35	1,4

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 0{,}63 + 1{,}68 + 1{,}44 = 3{,}8;$$

$$K_{K2} = 0{,}63 + 1{,}68 + 1{,}4 = 3{,}76.$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =90 людино-днів. Поправочний коефіцієнт,

який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.8. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 90 \cdot 1.8 \cdot 0.9 = 145,8 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =30 людино-днів, $К_П$ =0.8, $К_{СК}$ = 1, $К_{СТ}$ =0.9:

$$Т_2 = 30 \cdot 0.8 \cdot 0.9 = 21.6 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (145,8 + 21,6 + 4,8 + 21,6) \cdot 8 = 1550,4 \text{ людино-годин.}$$

$$Т_{II} = (145,8 + 21,6 + 6,91 + 21,6) \cdot 8 = 1567,28 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 20000 грн., один аналітик в області даних з окладом 23000. Визначимо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн.}, \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{20000 + 20000 + 23000}{3 \cdot 21 \cdot 8} = 125 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д, \quad (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 125 \cdot 1550{,}4 \cdot 1.2 = 232560$ грн.

II. $С_{ЗП} = 125 \cdot 1567{,}28 \cdot 1.2 = 235092$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 232560 \cdot 0.22 = 51163{,}4$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 235092 \cdot 0.22 = 51720{,}24$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 20000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 20000 \cdot 0,2 = 48000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 48000 \cdot (1 + 0.2) = 57600 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 57600 \cdot 0,22 = 12672 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 24000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.1 \cdot 0.25 \cdot 24000 = 6600 \text{ грн.,}$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.1 \cdot 24000 \cdot 0.06 = 1584 \text{ грн.,}$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.8 =$$

$$= 1491,2 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1491,2 \cdot 0,2 \cdot 4,87 \cdot 1,44 = 2091,5 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 24000 \cdot 0,67 = 16080 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \qquad (4.17)$$

$$C_{ЕКС} = 57600 + 12672 + 6600 + 1584 + 2091,5 + 16080 = 96627,5 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{М\text{-}Г} = C_{ЕКС} / T_{ЕФ} = 96627,5 / 1491,2 = 64,8 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_М = C_{М-Г} \cdot T, \quad (4.18)$$

I. $C_М = 64,8 \cdot 1550,4 = 100465,9$ грн.

II. $C_М = 64,8 \cdot 1567,28 = 101559,7$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} \cdot 0,67, \quad (4.19)$$

I. $C_Н = 232560 \cdot 0,67 = 155815,2$ грн.

II. $C_Н = 235092 \cdot 0,67 = 157511,64$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н, \quad (4.20)$$

I. $С_{ПП}$ = 232560 + 51163,4 + 100465,9 + 155815,2 = 540004,5 грн.

II. $С_{ПП}$ = 235092 + 51720,24 + 101559,7 + 157511,64 = 543883,58 грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$$К_{ТЕР1} = 3,8/540004,5 = 7,036 \cdot 10^{-6},$$

$$К_{ТЕР2} = 3,76/543883,58 = 6,913 \cdot 10^{-6}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}$= 7,036 $\cdot 10^{-6}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР}$ = 7,036 $\cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

– Вибір мови програмування – Python;
– Вибір середовища розробки – Google Colab;

– Вибір бібліотеки для роботи з нейронними мережами - TensorFlow.

Висновки до розділу 4

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

Однією з найбільш актуальних проблем класифікації зображень є класифікація медичних зображень, оскільки її розв’язання дало би можливість автоматизувати процес діагностування відповідних захворювань, що в свою чергу відкривало би перед нами багато цікавих перспектив. Одним із типів таких зображень є знімки очного дна людини, які робляться за допомогою спеціального приладу, що називається фундус-камерою. Очі без усяких сумнівів є одним із важливіших органів почуттів людини, і тому було би дуже корисно отримати можливість автоматизованого діагностування їх здоров’я, хоча б – найбільш поширені з них, а саме – катаракти та діабетичної ретинопатії.

Зараз найкращим засобом для розв’язання задачі класифікації зображень є застосування глибоких згорткових нейронних мереж, зокрема – заздалегідь навчені моделі, використовуючи переносне навчання.

Під час проведення дослідження було розглянуто необхідні дані як з предметної області, так і з теоретичних аспектів нейронних мереж, зокрема – основні складові персептронів та згорткових нейронних мереж, принципи їх роботи, деякі приклади архітектур нейронних мереж, а також методи оцінки і покращення роботи нейронних мереж, як то зменшення перенавчання та недонавчання або збільшення навчальних даних.

На базі розглянутих моделей та підходів, в третьому розділі було проведено експериментальну частину, в результаті якої було побудовано декілька моделей нейронних мереж на мові програмування Python. Підсумовуючи отримані результати, можна прийти для висновку, що найкращою моделлю для виявлення катаракти є нейронна мережа, побудована на базі VGG-16, оскільки показала однаково доволі непогані результати як на

валідаційній, так і на тестовій вибірці. У випадку діабетичної ретинопатії, найкращою моделлю виявилась побудована на базі VGG-19.

В подальшому, побудовані моделі нейронних мереж можна було би використовувати у програмних додатках, призначених для виявлення відповідних захворювань, зокрема – можна побудувати аналогічним принципом нейронні мережі для виявлення інших, менш поширених захворювань, та використовувати їх в поєднанні для діагностування загального здоров’я очей пацієнта.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Eye care, vision impairment and blindness. URL: https://www.who.int/health-topics/blindness-and-vision-loss#tab=tab_1 (Last accessed: 18.05.2023)
2. How to Protect Eyes from Mobile and Computer Screens. URL: https://kraffeye.com/blog/how-to-protect-eyes-from-mobile-and-computer-screens (Last accessed: 17.05.2023)
3. Очне дно. Медична енциклопедія. URL: https://medical-enc.com.ua/glaznoe_dno.htm (Last accessed: 17.05.2023)
4. Ретинальная (Фундус) камера – що це, коли застосовується, методика дослідження. URL: https://jak.koshachek.com/articles/retinalnaja-fundus-kamera-shho-ce-koli.html (Last accessed: 17.05.2023)
5. Common Eye Disorders and Diseases. URL: https://www.cdc.gov/visionhealth/basics/ced/index.html (Last accessed: 17.05.2023)
6. Muhammad Moshin Butt, D.N.E. Awang Iskandar, Sherif E. Abdelhamid, Ghazanfar Latif, Runna Alghazo. Diabetic Retinopathy Detection from Fundus Images of the Eye Using Hybrid Deep Learning Features, 2022. DOI: https://doi.org/10.3390/diagnostics12071607
7. Deepak Raj. Single-layer Neural Networks in Machine Learning (Perceptrons). URL: https://dev.to/codeperfectplus/single-layer-neural-networks-in-machine-learning-perceptrons-18n8 (Last accessed: 17.05.2023)
8. Naveen. What is multilayer perceptron? URL: https://www.nomidl.com/natural-language-processing/what-is-multilayer-perceptron/ (Last accessed: 18.05.2023)
9. Milos Simic. Adam Optimizer. URL: https://www.baeldung.com/cs/adam-optimizer (Last accessed: 17.05.2023)

10. Pengfei Nie. Convolution operation on images. URL: https://pengfeinie.github.io/convolution-operation-on-images/ (Last accessed: 17.05.2023)
11. Mahitha Kumar. How Padding helps in CNN? URL: https://www.numpyninja.com/post/how-padding-helps-in-cnn (Last accessed: 17.05.2023)
12. Debasish Kalita. Basics of CNN in Deep Learning. URL: https://www.analyticsvidhya.com/blog/2022/03/basics-of-cnn-in-deep-learning/ (Last accessed: 17.05.2023)
13. Vishal Rajput. Pooling layers in neural nets and their variants. URL: https://medium.com/aiguys/pooling-layers-in-neural-nets-and-their-variants-f6129fc4628b (Last accessed: 19.05.2023)
14. Jeremy Jordan. Common architectures in convolutional neural networks. URL: https://www.jeremyjordan.me/convnet-architectures/#resnet (Last accessed: 17.05.2023)
15. Aditi Rastogi. ResNet50. URL: https://blog.devgenius.io/resnet50-6b42934db431 (Last accessed: 17.05.2023)
16. Sumeet Kumat Agrawal. Metrics to Evaluate your Classification Model to take the right decisions. URL: https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/ (Last accessed: 17.05.2023)
17. Dewang Nautiyal. ML | Underfitting and Overfitting. URL: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/ (Last accessed: 17.05.2023)
18. Riccardo Andreoni. Regularization Techniques for Neural Networks. URL: https://towardsdatascience.com/regularization-techniques-for-neural-networks-379f5b4c9ac3 (Last accessed: 17.05.2023)
19. Saulo Baretto. Data Augmentation. URL: https://www.baeldung.com/cs/ml-data-augmentation (Last accessed: 17.05.2023)

20. Sanidhya Singh. Data Augmentation to solve imbalanced training data for Image Classification. URL: https://medium.com/an alytics-vidhya/data-augmentation-to-solve-imbalanced-training-data-for-image-classification-f6d888cbd596 (Last accessed: 18.05.2023)
21. Huazhu Fu, Jun Cheng, Yanwu Xu, Jiang Kiu. Glaucoma Detection based on Deep Learning Nwtwork in Fundus image, 2019. DOI: https://doi.org/10.1007/978-3-030-13969-8_6
22. Wejdan L. Alyoubi. Diabetic Retinopathy Fundus Image Classification and Lesions Localization System Using Deep Learning, 2021. DOI: https://doi.org/10.3390/s21113704

ДОДАТОК А
КОДИ ПРОГРАМ

Код програми, що використовувалась при попередньому розподіленні зображень по директоріях.

```
import numpy as np
import pandas as pd
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt
import shutil
import os
from google.colab import drive
drive.mount('/content/drive')
df=pd.read_excel('/content/drive/MyDrive/Data/data.xlsx')
display(df)
data=[]
for i in df['ID'].values:
  for x in ['Left','Right']:
    N,D,G,C,A,H,M,O=0,0,0,0,0,0,0,0
    if df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0]=='normal
fundus':
      N=1
    if 'retinopathy' in df.loc[df['ID']==i,x+'-Diagnostic
Keywords'].values[0] and ('hypertensive retinopathy' not in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] and 'myopia
retinopathy' not in df.loc[df['ID']==i,x+'-Diagnostic
Keywords'].values[0]  and 'myopic retinopathy' not in df.loc[df['ID']==i,x+'-
Diagnostic Keywords'].values[0]):
      D=1
    if 'glaucoma' in df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0]:
      G=1
    if 'cataract' in df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0]:
      C=1
    if 'age-related' in df.loc[df['ID']==i,x+'-Diagnostic
Keywords'].values[0]:
      A=1
    if 'hyperten' in df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0]:
      H=1
    if 'myop' in df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0]:
      M=1
    if 'laser' in df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or
'membran' in df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or
'pigmentation' in df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or
'drusen' in df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or
'detachment' in df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or
'atrophy' in df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'dust'
```

```
in df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'vitreous' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'coloboma' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'opacity' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'fibers' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'hole' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'pigmentosa' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'fold' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'occlusion' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'opacity' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'vessel' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'tessell' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'oil' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'old' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0] or 'hemorrhage' in
df.loc[df['ID']==i,x+'-Diagnostic Keywords'].values[0]:
      O=1
    if N+D+G+C+A+H+M+O!=0:
      data.append([f'{i}'+'_'+x.lower()+'.jpg',N,D,G,C,A,H,M,O])
    if 'no fundus image' in df.loc[df['ID']==i,x+'-Diagnostic Keywords']:
      data.pop(-1)
df_=pd.DataFrame(columns=['Image_name','N','D','G','C','A','H','M','O'],data=
data)
display(df_)
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Cataract distribution')
ax.hist(df_['C'][df_['C']==0],color='c')
ax.hist(df_['C'][df_['C']==1],color='r')
plt.xticks([0,1])
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Diabetes distribution')
ax.hist(df_['D'][df_['D']==0],color='c')
ax.hist(df_['D'][df_['D']==1],color='r')
plt.xticks([0,1])
df=pd.read_csv('/content/drive/MyDrive/Data/train.csv')
display(df)
path='/content/drive/MyDrive/Data/Images/'
i=0
for image in df[df['diagnosis']==0]['id_code'].values:
  try:
    shutil.move('/content/drive/MyDrive/Data/DR_/'+image+'.png','/content/dri
ve/MyDrive/Datasets/D/0'+image+'.png')
  except:
    i+=1
for image in df[df['diagnosis']!=0]['id_code'].values:
  try:
```

```
    shutil.move('/content/drive/MyDrive/Data/DR_/'+image+'.png','/content/dri
ve/MyDrive/Datasets/D/1'+image+'.png')
  except:
    i+=1
print(i)
path='/content/drive/MyDrive/Data/Images/'
i=0
for image in df_[df_['C']==1]['Image_name'].values:
  try:
    shutil.move(path+image,'/content/drive/MyDrive/Datasets/C/1/'+image)
  except:
    i+=1
print(i)
for image in df_[df_['C']==0]['Image_name'].values:
  try:
    shutil.move(path+image,'/content/drive/MyDrive/Datasets/C/0/'+image)
  except:
    i+=1
print(i)
for image in df_[df_['D']==1]['Image_name'].values:
  try:
    shutil.move(path+image,'/content/drive/MyDrive/Datasets/D/1/'+image)
  except:
    i+=1
print(i)
for image in df_[df_['D']==0]['Image_name'].values:
  try:
    shutil.move(path+image,'/content/drive/MyDrive/Datasets/D/0/'+image)
  except:
    i+=1
print(i)
```

Код програми, що використовувалась у випадку катаракти.

```
import numpy as np
import pandas as pd
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
import seaborn as sns
import shutil
import os
from google.colab import drive
drive.mount('/content/drive')
img_size=256
batch_size=128
def delete_gray(img,tol=7):
  if img.ndim==2:
    mask=img>tol
    return img[np.ix_(mask.any(1),mask.any(0))]
  elif img.ndim==3:
    gray_img=cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
    mask=gray_img>tol
    check_shape=img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0]
    if (check_shape==0):
      return img
    else:
      img1=img[:,:,0][np.ix_(mask.any(1),mask.any(0))]
      img2=img[:,:,1][np.ix_(mask.any(1),mask.any(0))]
      img3=img[:,:,2][np.ix_(mask.any(1),mask.any(0))]
      img=np.stack([img1,img2,img3],axis=-1)
    return img
def preprocess_image(image,sigmaX=10):
  image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
  image=delete_gray(image)
  image=cv2.resize(image,(img_size,img_size))
  image=cv2.addWeighted(image,4,cv2.GaussianBlur(image,(0,0),sigmaX),-4,128)
  return image
dist=np.random.choice(['train','val','test'],p=[0.8,0.1,0.1],size=7008)
X_train=np.empty(shape=(10*dist[dist=='train'].shape[0],img_size,img_size,3),
dtype='uint8')
X_val=np.empty(shape=(dist[dist=='val'].shape[0],img_size,img_size,3),dtype='
uint8')
X_test=np.empty(shape=(dist[dist=='test'].shape[0],img_size,img_size,3),dtype
='uint8')
y_train=[]
y_val=[]
y_test=[]
tr_i=0
```

```
vl_i=0
ts_i=0
i=0
mean = 0
stddev = 60
l1=os.listdir('/content/drive/MyDrive/Data/Datasets/C/1')
for img in l1:
  if dist[i]=='train':
    image=cv2.imread('/content/drive/MyDrive/Data/Datasets/C/1/'+img)
    image=cv2.resize(image,(img_size,img_size))
    X_train[tr_i]=image
    y_train.append(1)
    noise = np.zeros(image.shape, np.uint8)
    cv2.randn(noise, mean, stddev)
    n_image=cv2.add(image,noise)
    X_train[tr_i+1]=n_image
    y_train.append(1)
    h_image=cv2.flip(image,1)
    X_train[tr_i+2]=h_image
    y_train.append(1)
    v_image=cv2.flip(image,0)
    X_train[tr_i+3]=v_image
    y_train.append(1)
    i+=1
    tr_i+=4
  elif dist[i]=='val':
    image=cv2.imread('/content/drive/MyDrive/Data/Datasets/C/1/'+img)
    image=cv2.resize(image,(img_size,img_size))
    X_val[vl_i]=image
    y_val.append(1)
    i+=1
    vl_i+=1
  elif dist[i]=='test':
    image=cv2.imread('/content/drive/MyDrive/Data/Datasets/C/1/'+img)
    image=cv2.resize(image,(img_size,img_size))
    X_test[ts_i]=image
    y_test.append(1)
    i+=1
    ts_i+=1
l2=os.listdir('/content/drive/MyDrive/Data/Datasets/C/0')
for img in l2:
  if dist[i]=='train':
    image=cv2.imread('/content/drive/MyDrive/Data/Datasets/C/0/'+img)
    image=cv2.resize(image,(img_size,img_size))
    X_train[tr_i]=image
    y_train.append(0)
    noise = np.zeros(image.shape, np.uint8)
    cv2.randn(noise, mean, stddev)
```

```
    n_image=cv2.add(image,noise)
    X_train[tr_i+1]=n_image
    y_train.append(0)
    h_image=cv2.flip(image,1)
    X_train[tr_i+2]=h_image
    y_train.append(0)
    v_image=cv2.flip(image,0)
    X_train[tr_i+3]=v_image
    y_train.append(0)
    i+=1
    tr_i+=4
  elif dist[i]=='val':
    image=cv2.imread('/content/drive/MyDrive/Data/Datasets/C/0/'+img)
    image=cv2.resize(image,(img_size,img_size))
    X_val[vl_i]=image
    y_val.append(0)
    i+=1
    vl_i+=1
  elif dist[i]=='test':
    image=cv2.imread('/content/drive/MyDrive/Data/Datasets/C/0/'+img)
    image=cv2.resize(image,(img_size,img_size))
    X_test[ts_i]=image
    y_test.append(0)
    i+=1
    ts_i+=1
X_train=X_train[:tr_i]
y_train=y_train[:tr_i]
X_train, y_train = shuffle(X_train, y_train, random_state=0)
print(y_train)
print(X_train.shape[0])
print(X_val.shape[0])
print(X_test.shape[0])
print(y_train)
print(y_val)
print(y_test)
X_train=tf.convert_to_tensor(X_train)
y_train=tf.convert_to_tensor(y_train)
X_val=tf.convert_to_tensor(X_val)
y_val=tf.convert_to_tensor(y_val)
X_test=tf.convert_to_tensor(X_test)
y_test=tf.convert_to_tensor(y_test)
train_data=tf.data.Dataset.from_tensor_slices((X_train,y_train)).batch(batch_
size)
val_data=tf.data.Dataset.from_tensor_slices((X_val,y_val)).batch(batch_size)
test_data=tf.data.Dataset.from_tensor_slices((X_test,y_test)).batch(batch_siz
e)
base_model=tf.keras.applications.vgg16.VGG16(weights='imagenet',include_top=F
alse,input_shape=(img_size,img_size,3))
```

```
base_model.trainable=False
base_model.summary()
model=tf.keras.Sequential([base_model,
                     tf.keras.layers.Flatten(),
                     tf.keras.layers.Dropout(0.4),
                     tf.keras.layers.Dense(units=64,activation='leaky_relu'),
                     tf.keras.layers.Dense(units=4,activation='leaky_relu'),
                     tf.keras.layers.Dense(units=1,activation='sigmoid')])
model.summary()
model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['Precision
','Recall','Accuracy'])
es=tf.keras.callbacks.EarlyStopping(monitor='loss',min_delta=0.01,patience=1,
restore_best_weights=True)
hist=model.fit(train_data,epochs=4,validation_data=val_data,callbacks=[es])
model.save('/content/drive/MyDrive/Models/my_model_2_1')
print(model.evaluate(test_data))
y_test_pred=model.predict(X_test)
y_test_pred=[1*(y[0]>0.5) for y in y_test_pred]
cm=confusion_matrix(y_test,y_test_pred)
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Confusion matrix',fontsize=20)
sns.heatmap(cm,annot=True,cbar=True,fmt='d',cmap='Greens')
ax.set_xlabel('predicted label')
ax.set_ylabel('true label')
cmn=[x/sum(x) for x in cm]
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Normalized confusion matrix',fontsize=20)
sns.heatmap(cmn,annot=True,cbar=True,fmt='.3f',cmap='Greens')
ax.set_xlabel('predicted label')
ax.set_ylabel('true label')
plt.show()
base_model=tf.keras.applications.vgg19.VGG19(weights='imagenet',include_top=F
alse,input_shape=(img_size,img_size,3))
for l in base_model.layers[:-1]:
  l.trainable=False
base_model.summary()
model=tf.keras.Sequential([base_model,
                     tf.keras.layers.Flatten(),
                     tf.keras.layers.Dropout(0.4),
                     tf.keras.layers.Dense(units=64,activation='leaky_relu'),
                     tf.keras.layers.Dense(units=4,activation='leaky_relu'),
                     tf.keras.layers.Dense(units=1,activation='sigmoid')])
model.summary()
model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['Precision
','Recall','Accuracy'])
```

```
es=tf.keras.callbacks.EarlyStopping(monitor='loss',min_delta=0.001,patience=1
,restore_best_weights=True)
hist=model.fit(train_data,epochs=4,validation_data=val_data,callbacks=[es])
model.save('/content/drive/MyDrive/Models/my_model_2_2')
print(model.evaluate(test_data))
y_test_pred=model.predict(X_test)
y_test_pred=[1*(y[0]>0.5) for y in y_test_pred]
cm=confusion_matrix(y_test,y_test_pred)
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Confusion matrix',fontsize=20)
sns.heatmap(cm,annot=True,cbar=True,fmt='d',cmap='Greens')
ax.set_xlabel('predicted label')
ax.set_ylabel('true label')
cmn=[x/sum(x) for x in cm]
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Normalized confusion matrix',fontsize=20)
sns.heatmap(cmn,annot=True,cbar=True,fmt='.3f',cmap='Greens')
ax.set_xlabel('predicted label')
ax.set_ylabel('true label')
plt.show()
base_model=tf.keras.applications.resnet.ResNet50(weights='imagenet',include_t
op=False,input_shape=(img_size,img_size,3))
for l in base_model.layers[:-1]:
  l.trainable=False
base_model.summary()
model=tf.keras.Sequential([base_model,
                     tf.keras.layers.Flatten(),
                     tf.keras.layers.Dropout(0.4),
                     tf.keras.layers.Dense(units=64,activation='leaky_relu'),
                     tf.keras.layers.Dense(units=4,activation='leaky_relu'),
                     tf.keras.layers.Dense(units=1,activation='sigmoid')])
model.summary()
model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['Precision
','Recall','Accuracy'])
es=tf.keras.callbacks.EarlyStopping(monitor='loss',min_delta=0.001,patience=1
,restore_best_weights=True)
hist=model.fit(train_data,epochs=4,validation_data=val_data,callbacks=[es])
model.save('/content/drive/MyDrive/Models/my_model_2_3')
print(model.evaluate(test_data))
y_test_pred=model.predict(X_test)
y_test_pred=[1*(y[0]>0.5) for y in y_test_pred]
cm=confusion_matrix(y_test,y_test_pred)
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Confusion matrix',fontsize=20)
sns.heatmap(cm,annot=True,cbar=True,fmt='d',cmap='Greens')
```

```
ax.set_xlabel('predicted label')
ax.set_ylabel('true label')
cmn=[x/sum(x) for x in cm]
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Normalized confusion matrix',fontsize=20)
sns.heatmap(cmn,annot=True,cbar=True,fmt='.3f',cmap='Greens')
ax.set_xlabel('predicted label')
ax.set_ylabel('true label')
plt.show()
```

Код програми, що використовувалась у випадку діабетичної ретинопатії.

```
import numpy as np
import pandas as pd
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
import seaborn as sns
import shutil
import os
from google.colab import drive
drive.mount('/content/drive')
img_size=256
batch_size=256
def delete_gray(img,tol=7):
  if img.ndim==2:
    mask=img>tol
    return img[np.ix_(mask.any(1),mask.any(0))]
  elif img.ndim==3:
    gray_img=cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
    mask=gray_img>tol
    check_shape=img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0]
    if (check_shape==0):
      return img
    else:
      img1=img[:,:,0][np.ix_(mask.any(1),mask.any(0))]
      img2=img[:,:,1][np.ix_(mask.any(1),mask.any(0))]
      img3=img[:,:,2][np.ix_(mask.any(1),mask.any(0))]
      img=np.stack([img1,img2,img3],axis=-1)
    return img
def preprocess_image(image,sigmaX=10):
  image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
  image=delete_gray(image)
  image=cv2.resize(image,(img_size,img_size))
  image=cv2.addWeighted(image,4,cv2.GaussianBlur(image,(0,0),sigmaX),-4,128)
  return image
dist=np.random.choice(['train','val','test'],p=[0.8,0.1,0.1],size=8574)
print(dist[dist=='train'].shape[0],dist[dist=='val'].shape[0],dist[dist=='tes
t'].shape[0])
X_train=np.empty(shape=(10*dist[dist=='train'].shape[0],img_size,img_size,3),
dtype='uint8')
X_val=np.empty(shape=(dist[dist=='val'].shape[0],img_size,img_size,3),dtype='
uint8')
X_test=np.empty(shape=(dist[dist=='test'].shape[0],img_size,img_size,3),dtype
='uint8')
y_train=[]
y_val=[]
```

```
y_test=[]
tr_i=0
vl_i=0
ts_i=0
i=0
mean = 0
stddev = 60
l1=os.listdir('/content/drive/MyDrive/Data/Datasets/DR/1')
for img in l1:
  if dist[i]=='train':
    image=cv2.imread('/content/drive/MyDrive/Data/Datasets/DR/1/'+img)
    image=cv2.resize(image,(img_size,img_size))
    X_train[tr_i]=image
    y_train.append(1)
    h_image=cv2.flip(image,1)
    X_train[tr_i+1]=h_image
    y_train.append(1)
    v_image=cv2.flip(image,0)
    X_train[tr_i+2]=v_image
    y_train.append(1)
    noise = np.zeros(image.shape, np.uint8)
    cv2.randn(noise, mean, stddev)
    n_image=cv2.add(image,noise)
    X_train[tr_i+3]=n_image
    y_train.append(1)
    noise = np.zeros(h_image.shape, np.uint8)
    cv2.randn(noise, mean, stddev)
    hn_image=cv2.add(h_image,noise)
    X_train[tr_i+4]=hn_image
    y_train.append(1)
    noise = np.zeros(v_image.shape, np.uint8)
    cv2.randn(noise, mean, stddev)
    vn_image=cv2.add(v_image,noise)
    X_train[tr_i+5]=vn_image
    y_train.append(1)
    i+=1
    tr_i+=6
  elif dist[i]=='val':
    image=cv2.imread('/content/drive/MyDrive/Data/Datasets/DR/1/'+img)
    image=cv2.resize(image,(img_size,img_size))
    X_val[vl_i]=image
    y_val.append(1)
    i+=1
    vl_i+=1
  elif dist[i]=='test':
    image=cv2.imread('/content/drive/MyDrive/Data/Datasets/DR/1/'+img)
    image=cv2.resize(image,(img_size,img_size))
    X_test[ts_i]=image
```

```
    y_test.append(1)
    i+=1
    ts_i+=1
l2=os.listdir('/content/drive/MyDrive/Data/Datasets/DR/0')
for img in l2:
  if dist[i]=='train':
    image=cv2.imread('/content/drive/MyDrive/Data/Datasets/DR/0/'+img)
    image=cv2.resize(image,(img_size,img_size))
    X_train[tr_i]=image
    y_train.append(0)
    h_image=cv2.flip(image,1)
    X_train[tr_i+1]=h_image
    y_train.append(0)
    """
    v_image=cv2.flip(image,0)
    X_train[tr_i+2]=v_image
    y_train.append(0)
    """
    noise = np.zeros(image.shape, np.uint8)
    cv2.randn(noise, mean, stddev)
    n_image=cv2.add(image,noise)
    X_train[tr_i+2]=n_image
    y_train.append(0)
    i+=1
    tr_i+=3
  elif dist[i]=='val':
    image=cv2.imread('/content/drive/MyDrive/Data/Datasets/DR/0/'+img)
    image=cv2.resize(image,(img_size,img_size))
    X_val[vl_i]=image
    y_val.append(0)
    i+=1
    vl_i+=1
  elif dist[i]=='test':
    image=cv2.imread('/content/drive/MyDrive/Data/Datasets/DR/0/'+img)
    image=cv2.resize(image,(img_size,img_size))
    X_test[ts_i]=image
    y_test.append(0)
    i+=1
    ts_i+=1
X_train=X_train[:tr_i]
y_train=y_train[:tr_i]
X_val=X_val[:vl_i]
y_val=y_val[:vl_i]
X_test=X_test[:ts_i]
y_test=y_test[:ts_i]
dist=np.random.choice([0,1],p=[0,1],size=X_train.shape[0])
train=np.empty(shape=(3*X_train.shape[0],256,256,3),dtype='uint8')
train[:X_train.shape[0]]=X_train
```

```
i=X_train.shape[0]
tr_i=X_train.shape[0]
for image in X_train[:]:
  if dist[i-X_train.shape[0]]==1:
    (h,w) = image.shape[:2]
    ang=np.random.randint(5,41,1)[0]
    rot=np.random.choice([0,1],p=[0.5,0.5],size=1)[0]
    rot_mat = cv2.getRotationMatrix2D((h/2,w/2), ang, rot)
    result = cv2.warpAffine(image, rot_mat, (h,w), flags=cv2.INTER_LINEAR)
    train[tr_i]=result
    y_train.append(y_train[i-X_train.shape[0]])
    tr_i+=1
  i+=1
i=2*X_train.shape[0]
tr_i=2*X_train.shape[0]
for image in X_train[:]:
  if dist[i-2*X_train.shape[0]]==1:
    (h,w) = image.shape[:2]
    ang=np.random.randint(5,41,1)[0]
    rot=np.random.choice([0,1],p=[0.5,0.5],size=1)[0]
    rot_mat = cv2.getRotationMatrix2D((h/2,w/2), ang, rot)
    result = cv2.warpAffine(image, rot_mat, (h,w), flags=cv2.INTER_LINEAR)
    train[tr_i]=result
    y_train.append(y_train[i-2*X_train.shape[0]])
    tr_i+=1
  i+=1
X_train=train[:tr_i]
y_train=y_train[:tr_i]
X_train, y_train = shuffle(X_train, y_train, random_state=0)
print(y_train)
print(X_train.shape[0])
print(X_val.shape[0])
print(X_test.shape[0])
print(y_train)
print(y_val)
print(y_test)
X_train=tf.convert_to_tensor(X_train)
y_train=tf.convert_to_tensor(y_train)
X_val=tf.convert_to_tensor(X_val)
y_val=tf.convert_to_tensor(y_val)
X_test=tf.convert_to_tensor(X_test)
y_test=tf.convert_to_tensor(y_test)
train_data=tf.data.Dataset.from_tensor_slices((X_train,y_train)).batch(batch_
size)
val_data=tf.data.Dataset.from_tensor_slices((X_val,y_val)).batch(batch_size)
test_data=tf.data.Dataset.from_tensor_slices((X_test,y_test)).batch(batch_siz
e)
```

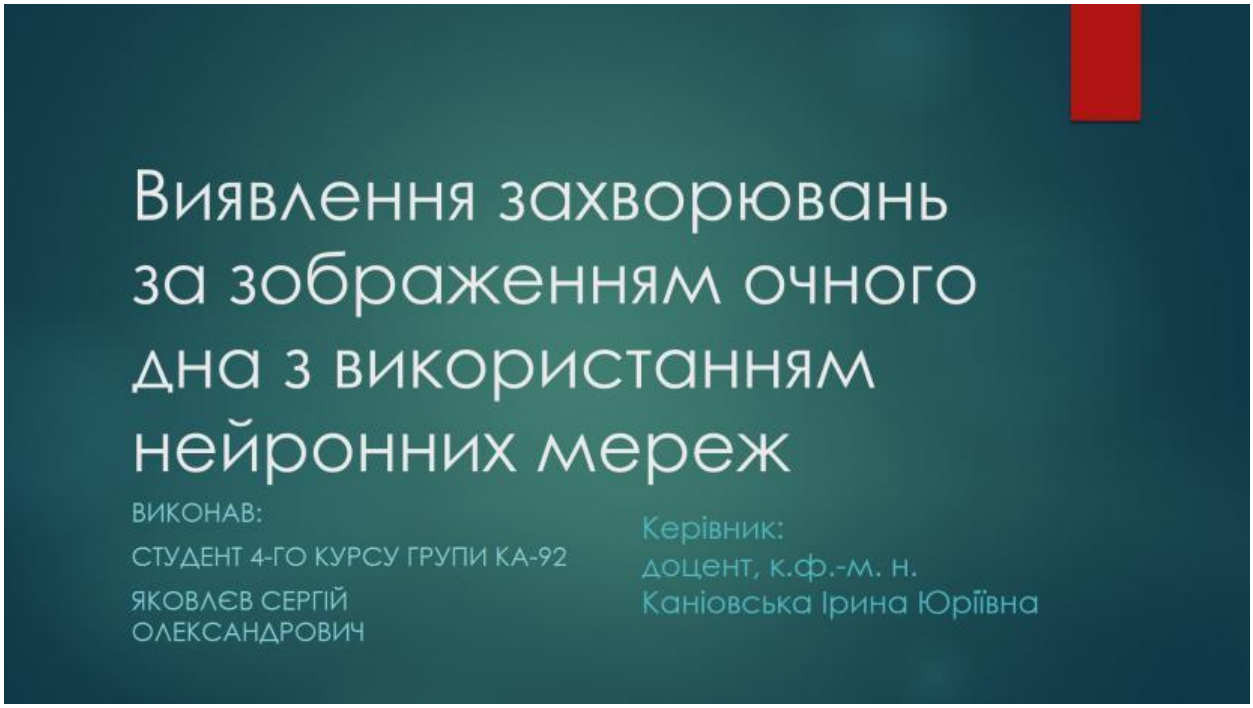
```
base_model=tf.keras.applications.resnet.ResNet50(weights='imagenet',include_t
op=False,input_shape=(img_size,img_size,3))
for l in base_model.layers[:-1]:
  l.trainable=False
base_model.summary()
model=tf.keras.Sequential([base_model,
                    tf.keras.layers.Flatten(),
                    tf.keras.layers.Dropout(0.5),
                    tf.keras.layers.Dense(units=256,activation='leaky_relu'),
                    tf.keras.layers.Dropout(0.25),
                    tf.keras.layers.Dense(units=32,activation='leaky_relu'),
                    tf.keras.layers.Dense(units=8,activation='leaky_relu'),
                    tf.keras.layers.Dense(units=1,activation='sigmoid')])
model.summary()
model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['Precision
','Recall','Accuracy'])
es=tf.keras.callbacks.EarlyStopping(monitor='loss',min_delta=0.005,patience=1
,restore_best_weights=True)
hist=model.fit(train_data,epochs=5,validation_data=val_data,callbacks=[es])
model.save('/content/drive/MyDrive/Models/my_model_1_3')
print(model.evaluate(test_data))
y_test_pred=model.predict(X_test)
y_test_pred=[1*(y[0]>0.5) for y in y_test_pred]
cm=confusion_matrix(y_test,y_test_pred)
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Confusion matrix',fontsize=20)
sns.heatmap(cm,annot=True,cbar=True,fmt='d',cmap='Greens')
ax.set_xlabel('predicted label')
ax.set_ylabel('true label')
cmn=[x/sum(x) for x in cm]
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Normalized confusion matrix',fontsize=20)
sns.heatmap(cmn,annot=True,cbar=True,fmt='.3f',cmap='Greens')
ax.set_xlabel('predicted label')
ax.set_ylabel('true label')
plt.show()
base_model=tf.keras.applications.vgg16.VGG16(weights='imagenet',include_top=F
alse,input_shape=(img_size,img_size,3))
for l in base_model.layers[:-1]:
  l.trainable=False
base_model.summary()
model=tf.keras.Sequential([base_model,
                    tf.keras.layers.Flatten(),
                    tf.keras.layers.Dropout(0.5),
                    tf.keras.layers.Dense(units=256,activation='leaky_relu'),
                    tf.keras.layers.Dropout(0.25),
```

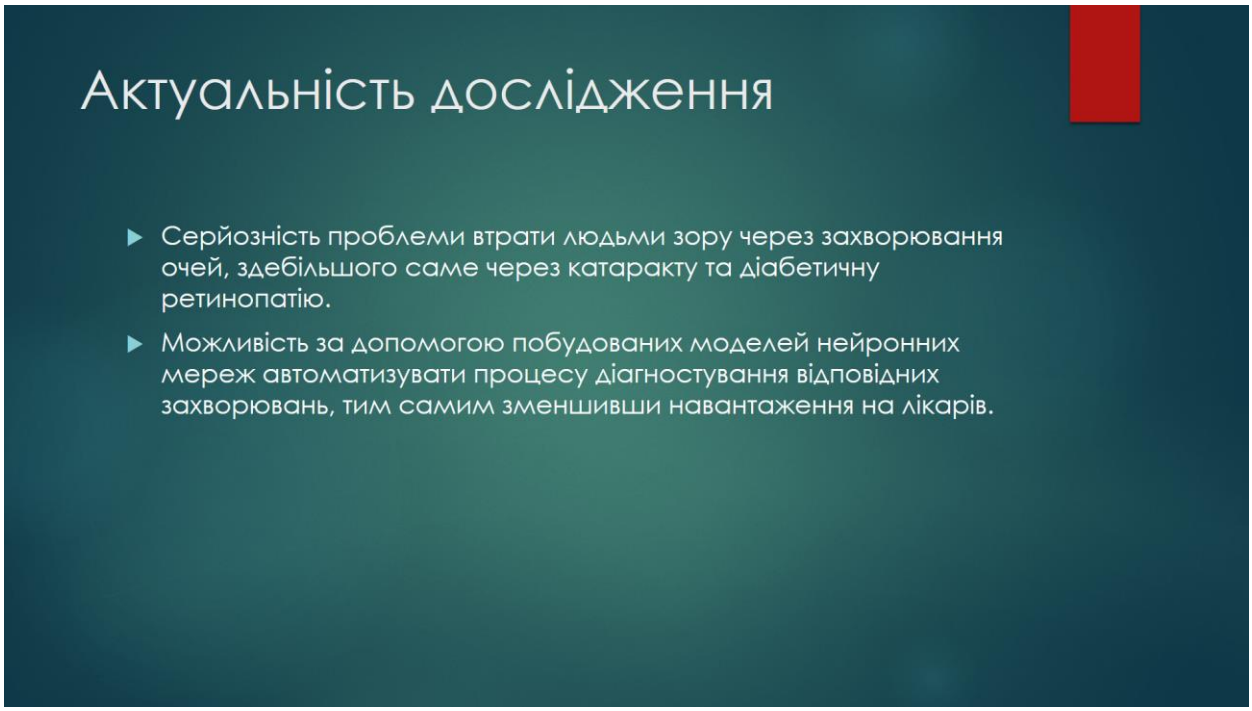
```
                    tf.keras.layers.Dense(units=32,activation='leaky_relu'),
                    tf.keras.layers.Dense(units=8,activation='leaky_relu'),
                    tf.keras.layers.Dense(units=1,activation='sigmoid')])
model.summary()
model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['Precision
','Recall','Accuracy'])
es=tf.keras.callbacks.EarlyStopping(monitor='loss',min_delta=0.01,patience=1,
restore_best_weights=True)
hist=model.fit(train_data,epochs=5,validation_data=val_data,callbacks=[es])
model.save('/content/drive/MyDrive/Models/my_model_1_1')
print(model.evaluate(test_data))
y_test_pred=model.predict(X_test)
y_test_pred=[1*(y[0]>0.5) for y in y_test_pred]
cm=confusion_matrix(y_test,y_test_pred)
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Confusion matrix',fontsize=20)
sns.heatmap(cm,annot=True,cbar=True,fmt='d',cmap='Greens')
ax.set_xlabel('predicted label')
ax.set_ylabel('true label')
cmn=[x/sum(x) for x in cm]
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Normalized confusion matrix',fontsize=20)
sns.heatmap(cmn,annot=True,cbar=True,fmt='.3f',cmap='Greens')
ax.set_xlabel('predicted label')
ax.set_ylabel('true label')
plt.show()
base_model=tf.keras.applications.vgg19.VGG19(weights='imagenet',include_top=F
alse,input_shape=(img_size,img_size,3))
for l in base_model.layers[:-1]:
  l.trainable=False
base_model.summary()
model=tf.keras.Sequential([base_model,
                    tf.keras.layers.Flatten(),
                    tf.keras.layers.Dropout(0.5),
                    tf.keras.layers.Dense(units=256,activation='leaky_relu'),
                    tf.keras.layers.Dropout(0.25),
                    tf.keras.layers.Dense(units=32,activation='leaky_relu'),
                    tf.keras.layers.Dense(units=8,activation='leaky_relu'),
                    tf.keras.layers.Dense(units=1,activation='sigmoid')])
model.summary()
model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['Precision
','Recall','Accuracy'])
es=tf.keras.callbacks.EarlyStopping(monitor='loss',min_delta=0.01,patience=1,
restore_best_weights=True)
hist=model.fit(train_data,epochs=5,validation_data=val_data,callbacks=[es])
model.save('/content/drive/MyDrive/Models/my_model_1_2')
```

```
print(model.evaluate(test_data))
y_test_pred=model.predict(X_test)
y_test_pred=[1*(y[0]>0.5) for y in y_test_pred]
cm=confusion_matrix(y_test,y_test_pred)
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Confusion matrix',fontsize=20)
sns.heatmap(cm,annot=True,cbar=True,fmt='d',cmap='Greens')
ax.set_xlabel('predicted label')
ax.set_ylabel('true label')
cmn=[x/sum(x) for x in cm]
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_title('Normalized confusion matrix',fontsize=20)
sns.heatmap(cmn,annot=True,cbar=True,fmt='.3f',cmap='Greens')
ax.set_xlabel('predicted label')
ax.set_ylabel('true label')
plt.show()
```

ДОДАТОК Б

ПРЕЗЕНТАЦІЯ





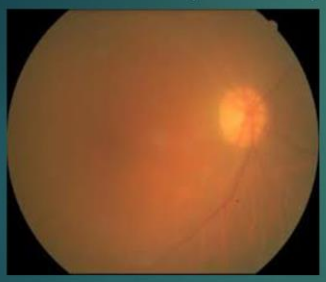
- Об'єкт дослідження – деякі захворювання очей
- Предмет дослідження – виявлення захворювань очей за зображенням очного дна за допомогою нейронних мереж
- Мета дослідження - проаналізувати існуючи методи класифікації зображень, побудувати на їх основі моделі для виявлення катаракти та діабетичної ретинопатії у випадку, коли в якості альтернативи до них виступають знімки не тільки здорового ока, але і знімки з іншими захворюваннями, відмінними від них, можливо – кількома

Постановка задачі

- Розібрати основи нейронних мереж та їхнє застосування в задачі класифікації зображень, зокрема – медичних.
- Побудувати моделі нейронних мереж для виявлення двох найпоширеніших захворювань очей – катаракти та діабетичної ретинопатії у випадку, коли відсутність захворювання має на увазі не обов'язково здорове око, але і наявність інших захворювань, можливо – кількох.
- Порівняти отримані результати та зробити висновки щодо доцільності застосування таких моделей.

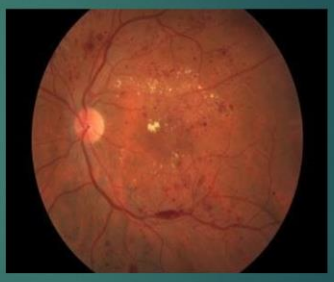
Катаракта та діабетична ретинопатія

Очне дно при катаракті



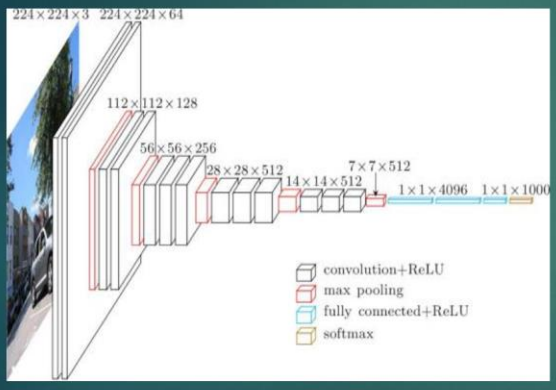
- Катаракта – це помутніння кришталика ока.
- Вона є головною причиною втрати зору у світі і при цьому може виникнути в будь-якому віці з різних причин, або бути присутньою від народження.

Очне дно при діабетичній ретинопатії



- Діабетична ретинопатія – це одне із ускладнень діабету.
- Дане захворювання проявляється у прогресуючому пошкодженні кровоносних судин сітківки, яка є світлочутливою тканиною в задній частині ока і необхідна для нормального зору.

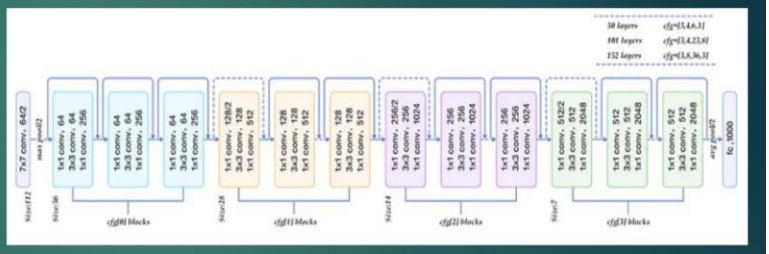
Архітектура моделей VGG-16 та VGG-19



- На малюнку зображено архітектуру згорткової нейронної мережі VGG-16.
- Модель VGG-19 має майже ту ж саму архітектуру, з відмінностями лишу у тому, що вона є трохи більшою, за рахунок додаткового шару згортки з 256 каналами, та по одному додатковому шару з 512 каналами в обох випадках.

Залишкові мережі

- Суть даного класу нейронних мереж полягає у тому, що перед тим, як застосувати функцію активації до значень, ми додаємо до них значення з минулого шару.
- Перевага їх застосування полягає в тому, що вони можуть надати додаткову інформацію, і при цьому, якщо вона є зайвою, можуть звестись до звичайних нейронних мереж.
- На малюнку зображані приклади моделей ResNet в залежності від кількості шарів.



Аналіз базових даних для навчання

- В якості базового набору даних використовувався набір даних Ocular Disease Detection з платформи Kaggle.
- Недоліком даного набору даних є зокрема те, що інформація в таблиці міститься в цілому по кожному пацієнту, а не окремо по кожному його оку.
- Проте, ця проблема вирішується пошуком по ключовим словам. В решті-решт, отримуємо файл представлений на рисунку справа.
- Зауважимо також, що літери у назвах колонок відповідають стану ока. Наприклад, N – Normal, D – Diabetic retinopathy, C-Cataract і т.д.

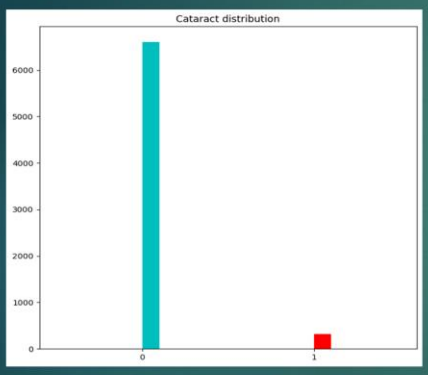
	ID	Patient Age	Patient Sex	Left-Fundus	Right-Fundus	Left-Diagnostic Keywords	Right-Diagnostic Keywords	N	D	G	C	A	H	M	O
0	0	69	Female	0_left.jpg	0_right.jpg	cataract	normal fundus	0	0	0	1	0	0	0	0
1	1	57	Male	1_left.jpg	1_right.jpg	normal fundus	normal fundus	1	0	0	0	0	0	0	0
2	2	42	Male	2_left.jpg	2_right.jpg	laser spot, moderate non proliferative retinopathy	moderate non proliferative retinopathy	0	1	0	0	0	0	0	1
3	3	66	Male	3_left.jpg	3_right.jpg	normal fundus	branch retinal artery occlusion	0	0	0	0	0	0	0	1
4	4	53	Male	4_left.jpg	4_right.jpg	macular epiretinal membrane	mild nonproliferative retinopathy	0	1	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3495	4686	63	Male	4686_left.jpg	4686_right.jpg	severe nonproliferative retinopathy	proliferative diabetic retinopathy	0	1	0	0	0	0	0	0
3496	4688	42	Male	4688_left.jpg	4688_right.jpg	moderate non proliferative retinopathy	moderate non proliferative retinopathy	0	1	0	0	0	0	0	0
3497	4689	54	Male	4689_left.jpg	4689_right.jpg	mild nonproliferative retinopathy	normal fundus	0	1	0	0	0	0	0	0
3498	4690	57	Male	4690_left.jpg	4690_right.jpg	mild nonproliferative retinopathy	mild nonproliferative retinopathy	0	1	0	0	0	0	0	0
3499	4784	58	Male	4784_left.jpg	4784_right.jpg	hypertensive retinopathy, age-related macular d...	hypertensive retinopathy, age-related macular d...	0	0	0	0	1	1	0	0

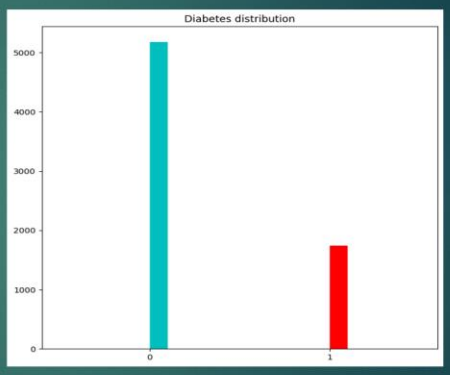
3500 rows × 15 columns

	Image_name	N	D	G	C	A	H	M	O
0	0_left.jpg	0	0	0	1	0	0	0	0
1	0_right.jpg	1	0	0	0	0	0	0	0
2	1_left.jpg	1	0	0	0	0	0	0	0
3	1_right.jpg	1	0	0	0	0	0	0	0
4	2_left.jpg	0	1	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...
6915	4689_right.jpg	1	0	0	0	0	0	0	0
6916	4690_left.jpg	0	1	0	0	0	0	0	0
6917	4690_right.jpg	0	1	0	0	0	0	0	0
6918	4784_left.jpg	0	0	0	0	1	1	0	0
6919	4784_right.jpg	0	0	0	0	1	1	0	0

6920 rows × 9 columns

Аналіз базових даних для навчання

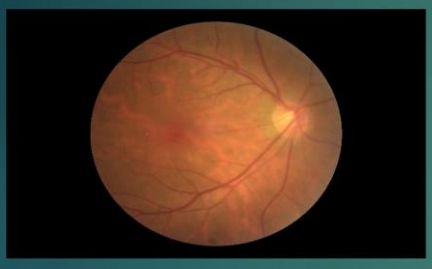




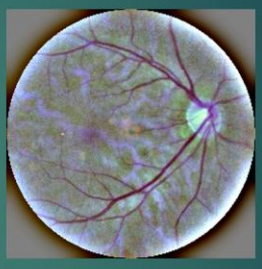
- Розподіл класів в базовому наборі даних для обох випадків зображено на рисунках.
- У випадку катаракти було лише застосовано збільшення всього навчального набору даних в 4 рази за рахунок деяких перетворень, інформацію про які буде більш детально наведено далі.
- У випадку діабетичної ретинопатії було перш за все поєднано наш набір даних з набором даних зі змагання APTOS 2019 Blindness Detection. При цьому було перевірено дані на сумісність та , про всяк випадок, на наявність дублікатів.

Попередня обробка зображень

До обробки



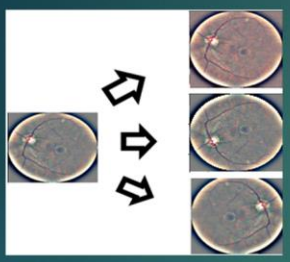
Після обробки

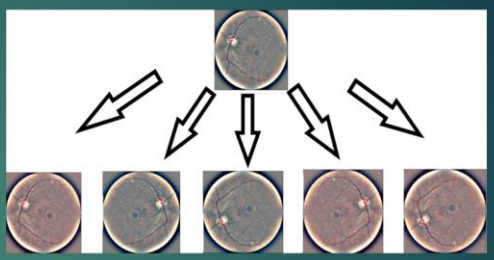


В якості попередньої обробки зображень було виконано наступне:

- Обрізано зайву чорну частину зображення по краях, яка не несе в собі жодної корисної інформації;
- Проведено роботу над насиченістю зображення, що вирішує проблему різного освітлення на знімках та підвищує чіткість зображення;
- Розмір усіх зображень змінено до спільного 256 на 256 пікселів.

Збільшення навчальних даних





- Перше зображення відповідає випадку катаракти, і наведена аугментація застосовувалась до зображень обох класів (горизонтальне і вертикальне віддзеркалювання та додавання випадкового шуму).
- Друге зображення стосується випадку діабетичної ретинопатії, та застосовувалось лише до зображень, що відповідають наявності наведеної хвороби, в той час, як до інших зображень застосовувалось спочатку лише горизонтальне віддзеркалювання та накладання випадкового шуму.
- Зауважимо також, що у другому випадку до всіх зображень потім ще двічі застосовувався випадковий поворот на кут від 5 до 40 градусів у випадковому напрямку.

Архітектури моделей

Випадок катаракти

```
model=tf.keras.Sequential([base_model,
                           tf.keras.layers.Flatten(),
                           tf.keras.layers.Dropout(0.4),
                           tf.keras.layers.Dense(units=64,activation='leaky_relu'),
                           tf.keras.layers.Dense(units=4,activation='leaky_relu'),
                           tf.keras.layers.Dense(units=1,activation='sigmoid')])
```

Випадок діабетичної ретинопатії

```
model=tf.keras.Sequential([base_model,
                           tf.keras.layers.Flatten(),
                           tf.keras.layers.Dropout(0.5),
                           tf.keras.layers.Dense(units=256,activation='leaky_relu'),
                           tf.keras.layers.Dropout(0.25),
                           tf.keras.layers.Dense(units=32,activation='leaky_relu'),
                           tf.keras.layers.Dense(units=8,activation='leaky_relu'),
                           tf.keras.layers.Dense(units=1,activation='sigmoid')])
```

- В якості базових моделей застосовуються моделі VGG-16, VGG-19, ResNet50 за допомогою переносного навчання без повно-зв'язних шарів.

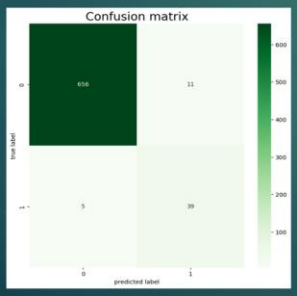
Метрики оцінки якості моделі бінарної класифікації

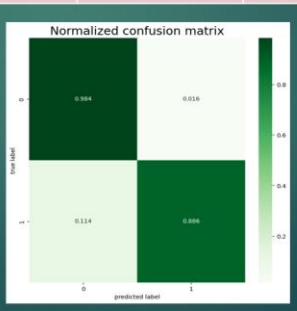
- Точність (англ. Accuracy): $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- Інша точність (англ. Precision): $Precision = \frac{TP}{TP+FP}$
- Повнота (англ. Recall): $\text{Recall} = \frac{TP}{TP+FN}$
- F1-міра (англ. F1-score): $\text{F1} = \frac{2}{\frac{1}{Precision}+\frac{1}{Recall}} = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$
- Матриця неточностей (англ. Confusion matrix):
$$\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix}$$
- Нормалізована матриця неточностей (англ. Normalized confusion matrix):
$$\begin{bmatrix} \frac{TN}{TN+FP} & \frac{FP}{TN+FP} \\ \frac{FN}{TP+FN} & \frac{TP}{TP+FN} \end{bmatrix}$$

- TP – кількість екземплярів класу 1, що класифіковані як екземпляр класу 1.
- TN – кількість екземпляру класу 0, що класифіковані як екземпляр класу 0.
- FP – кількість екземплярів класу 0, що класифіковані як екземпляр класу 1.
- FN – кількість екземплярів класу 1, що класифіковані як екземпляр класу 0.

Найкраща модель у випадку катаракти

Base model/metric	Accuracy	Precision	Recall	F1
VGG-16	0.977	0.78	0.886	0.83
VGG-19	0.977	0.868	0.75	0.805
ResNet50	0.975	0.861	0.704	0.774

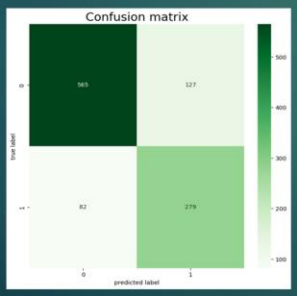


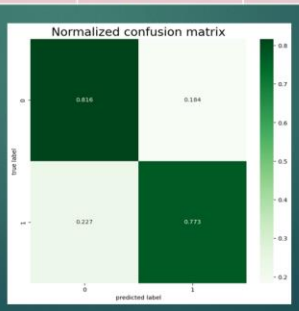


- У випадку виявлення катаракти найкращою за значенням F1-міри виявилась модель нейронних мереж, побудована на базі VGG-16.
- Для неї зокрема наведена матриця неточностей на тестовій вибірці та її нормалізований варіант.

Найкраща модель у випадку діабетичної ретинопатії

Base model/metric	Accuracy	Precision	Recall	F1
VGG-16	0.759	0.608	0.831	0.702
VGG-19	0.801	0.687	0.773	0.727
ResNet50	0.763	0.611	0.853	0.712





- У випадку виявлення діабетичної ретинопатії найкращою за значенням F1-міри виявилась модель нейронних мереж, побудована на базі VGG-19.
- Для неї зокрема наведена матриця неточностей на тестовій вибірці та її нормалізований варіант.

Результати роботи

- Було розглянуто предметну область.
- Було детально розглянуто основні теоретичні відомості нейронних мереж в задачі класифікації зображень.
- Побудовано моделі нейронних мереж для виявлення катаракти та діабетичної ретинопатії, перша з яких показала значення міри F1 рівне 0.83 на тестовій вибірці, в другому випадку отримали значення 0.727 відповідної метрики, що, за великим рахунком, є доволі непоганими результатами.

Подальші дослідження

- Побудува аналогічним чином моделей нейронних мереж для виявлення інших , менш поширених захворювань очей. Зокрема, розглянути випадки, де перед безпосередньо класифікацією бажано локалізувати на зображенні певну частину очного дна, за якою виконувати класифікацією буде простіше, відокремити цю частину, та вже за нею виконувати класифікацію.
- Навчання кращих моделей.
- Навчання моделей на більш якісному наборі даних.

Дякую за увагу!