Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування закладу вищої освіти)

Навчально-науковий інститут інформаційних технологій і робототехніки
(повне найменування інституту, назва факультету (відділення))

Кафедра автоматики, електроніки та телекомунікацій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до кваліфікаційної роботи

магістр
(ступінь вищої освіти)

на тему **РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ДОСЛІДЖЕННЯ МЕТОДІВ ОБРОБКИ ДАНИХ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ**

Виконав: студент 6 курсу, групи 601-ТТ
спеціальності 172 «Телекомунікації та
(шифр і назва напряму підготовки, спеціальності)
радіотехніка

Фенько В.В.
(прізвище та ініціали)
Керівник Косенко В.В.
(прізвище та ініціали)
Рецензент Шефер О.В.
(прізвище та ініціали)

Полтава – 2022 рік

Національний університет «Полтавська політехніка імені Юрія Кондратюка»
Інститут Навчально-науковий інститут інформаційних технологій і робототехніки
Кафедра Автоматики, електроніки та телекомунікацій
Ступінь вищої освіти Магістр
Спеціальність 172 «Телекомунікації та радіотехніка»

ЗАТВЕРДЖУЮ

Завідувач кафедри автоматики, електроніки та телекомунікацій

_____ О.В. Шефер
“ ___ ” _____ 2022 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Феньку Валентину Володимировичу

1. Тема проекту (роботи) **«Розроблення програмного забезпечення та дослідження методів обробки даних на основі нейронних мереж»**
керівник проекту (роботи) Косенко Віктор Васильович, д.т.н., професор
затверджена наказом вищого навчального закладу від “12 ” 08 2022 року № 544 фа

2. Строк подання студентом проекту (роботи) 21.11.2022 р.

3. Вихідні дані до проекту (роботи) Мова програмування C#, мультимедійна платформа Unity, Принципи функціонування нейронних мереж прямого поширення, Методи навчання нейронних мереж.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Принципи функціонування, побудови та особливості штучних нейронних мереж. Дослідження методів обробки інформаціїї на основі нейронних мереж. Розроблення програмного забезпечення. Вибір архітектуру, навчання та дослідження мережі у задачах стиснення зображень.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових плакатів):
1) Процес створення моделі нейронної мережі
2) Налаштування параметрів мережі
3) Керування навчанням моделі
4) Статистика навчання
5) Візуалізація роботи мережі
6) Класифікація та стиснення зображень
7) Bottle Neck архітектура

8) Модель мережі стиснення зображень

6. Дата видачі завдання 01.09.2022 р.

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів магістерської роботи	Термін виконання етапів роботи			Примітка (плакати)
1	Аналіз принципів функціонування нейронних мереж, їх особливостей, вибір напрямку дослідження.	13.09.22		15%	Пл. 1
2	Дослідження принципів функціонування згорткових нейронних мереж (CNN) та її основних компонентів.	27.09.22	I	30%	Пл. 2
3	Дослідження навчання нейронних мереж	10.10.22		40%	Пл. 3
4	Вибір методів та початок розроблення ПЗ	17.10.22		50 %	Пл. 4
5	Розроблення програмного забезпечення	25.10.22	II	60%	Пл. 5
6	Навчання мережі за допомогою ПЗ та збір статистики	07.11.22		70%	Пл. 6-8
7	Оформлення магістерської роботи	07.12.22	III	100%	

Магістрант _____ Фенько В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Косенко В.В.
(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ANN (англ. artifitial meural metwork) – штучна нейронна мережа.

CNN (англ. convolutional meural metwork) – згорткова нейронна мережа.

ReLU (англ. rectified linear unit) – основна функція активації CNN.

FCL (англ. fully connected layer) – повнозв’язний шар.

MLP (англ.multilayer perseptron) – багатошаровий перцептрон.

FCL (англ. fully connected layer) – повнозв’язний шар.

CPU (англ. central processor unit) – центральний процесор.

GPU (англ. graphic processor unit) – графічний процесор.

DCT (англ. discrete cosine transform) – дискретне косинусне перетворення

BP (англ. back propagation) – зворотне розповсюдження.

BN (англ. batch normalization) – пакетна нормалізація.

CE (англ. Cross Entropy) – крос ентропія.

ВСТУП

Розвиток у галузі дослідження нових методів обробки інформації на основі нейронних мереж пов’язаний із збільшенням кількісних характеристик досліджуваних мереж, а також із поліпшенням підходів щодо організації зв’язків нейронної мережі для різного типу задач. Спочатку нейронні мережі розглядалися здебільшого у якості рішення для задач, де звичайні алгоритми не давали прийнятного результату (машинній зір, задачі класифікації, прогнозування даних, генерація нового контенту тощо), але з часом з’явилися дослідження в бік використання штучного інтелекту для вирішення більш тривіальних задач.

Однією із таким задач є задача стискання графічних образів. Здатність згорткових нейронних мереж класифікувати зображення пов’язана із тим, що вони виокремлюють ознаки, що повторюються для різних зображень (патерни), а потім на їх основі роблять ймовірнісну оцінку належності зображення до того чи іншого класу. З підвищенням складності нейронних мереж виявилося, що даний підхід може бути використаний для того, щоб за деякими ознаками відтворить вихідне зображення. Стискання, при цьому, відбувається за рахунок того, що кількісна характеристика ознак (обсяг займаної ними пам’яті) менше за вихідне зображення.

При розв’язанні будь-якої задачі за допомогою нейронних мереж заздалегідь невідомі вихідні параметри мережа (топологія, оптимальні функції активації, метод навчання і т.і). Таким чином, розв’язання подібної задачі зводиться до емпіричного дослідження вибору архітектури мережі та параметрів для її навчання.

Розробка програмного забезпечення, що дозволяє виводити візуальну інформацію щодо навчання мережі (статистика, поведінка окремих ділянок мережі тощо) спрощує підбір параметрів мережі для оптимального вирішення поставленої задачі.

Тема даної магістерської роботи є актуальною, адже дослідження штучних нейронних мереж зараз набуває активного розвитку у зв’язку із підвищенням розрахункових можливостей сучасних комп’ютерів. Робота також є інноваційною,

адже програмного забезпечення із графічним інтерфейсом в якому можна створювати та навчати нейронні мережі небагато або воно має значні обмеження у функціоналі.

Метою роботи є розробка програмного забезпечення із графічним інтерфейсом, що дозволяє здійснювати навчання нейронних мереж прямого розповсюдження (згоркових та повнозв'язних) для роботи із зображеннями, а також дослідження навчання мережі стиснення даних.

Поставлену задачі можна розділити на наступні етапи:

1) Розроблення системи вводу/виводу програмного забезпечення. Сюди входить безпосередньо графічний інтерфейс (редагування топології мережі, керування навчанням, вивід статистики), а також файлова система застосунку (завантаження та зберігання налаштувань нейронної мережі, а також робота із датасетами).
2) Програмна реалізація основних компонентів згорткових нейронних мереж для вирішення задач класифікації та стискання зображень з можливістю їх навчання за допомогою алгоритму зворотного розповсюдження помилки.
3) Вибір топології, навчання та дослідження характеристик мережі у задачах стискання зображень.

1 ПРИНЦИПИ ФУНКЦІОНУВАННЯ, ПОБУДОВИ ТА ОСОБЛИВОСТІ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

1.1 Структура та принцип роботи нейронних мереж

1.1.1 Референсна модель побудови нейронних мереж

Людство, зокрема галузь інженерії, досить часто послуговується досягненнями природи для створення нових технічних рішень на їх основі. Штучні нейронні мережі не є виключенням - дослідження принципів функціонування біологічних нейронних мереж (власне, мозку людей та тварин) дозволило описати математичну модель яка стала основою у створенні нових напрямків комп'ютерного інтелекту.

Створення штучних нейронних мереж в значній мірі зобов'язано дослідженням у галузі нейробіології, а в окремих випадках робота штучної нейронної мережі може нагадувати в деякій мірі роботу мозку. Проте дуже часте порівняння процесів які відбувається у мозку та принципу закладеного у штучні мережі є недоцільним, адже реальний зв'язок між біологією та комп'ютерним інтелектом дуже слабкий, і часто викликає ряд непорозумінь. Крім того, наші знання про принципі функціонування біологічного мозку не є фундаментальними, що свідчить про неможливість відтворення закладених у мозок процесів у якості зрозумілої математичної моделі.

Навіть з огляду на вищесказане є корисним знати деякі основні положення щодо принципів роботи біологічної нервової системи, адже штучні нейронні мережі намагаються наблизитися по ефективності до біологічних мереж при рішення задач, які без проблем вирішують люди та тварини [1].

Дослідження штучних нейронних мереж, в загальному випадку, пов'язані із тим, що спосіб обробки інформації нервовою системою людини або тварини в значній мірі відрізняється від того, як це роблять звичайні цифрові комп'ютери.

Дозволяючи спрощення, мозок можна уявити як дуже складну, нелінійну структуру, основними компонентами якої є нейрони, що мають властивість

самоорганізації, яка в свою чергу дозволяє виконувати складні для звичайної обчислювальної техніки задачі. Прикладами можуть бути класифікація (розпізнавання образів, звуків, запахів та т.і.), моторні функції (звичайна для нас дія, наприклад ходьба – це задіяння величезної кількості м’язів у правильній послідовності, чим власне і займається мозок), обробка сигналів (світлових, звукових і т.і) [2].

Нервова система людини надзвичайно складна. По різним оцінкам головний мозок людини в середньому налічує десятки, а іноді пишуть і про сотні мільярдів нейронів та на порядки вищі значення кількості зв’язків між ними. Унікальна особливість нейрону полягає в тому, що він має змогу передавати, приймати та обробляти електрохімічні сигнали, що передаються нервовими зв’язками. Сукупність нейронів та їх зв’язків утворюють систему комунікації головного мозку людини.

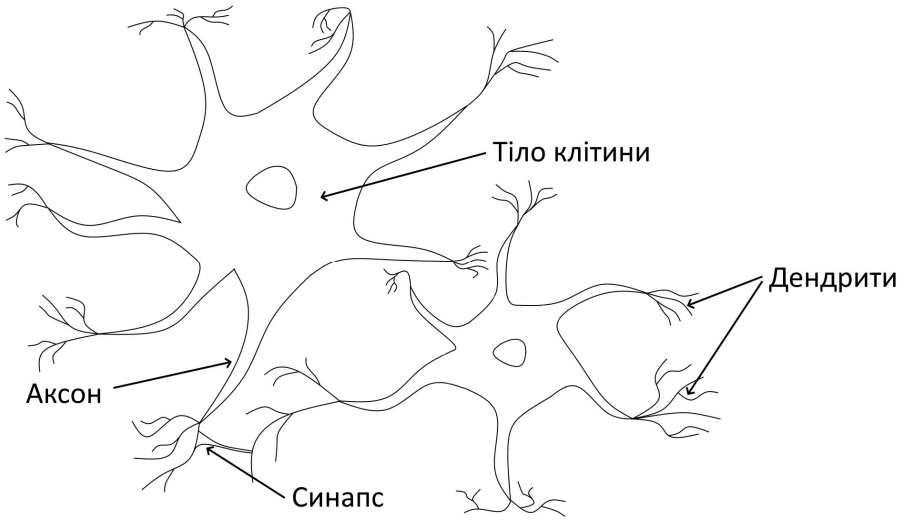


Рисунок 1.1 – Структура біологічного нейрона

На рисунку 1.1 зображена структура із двох біологічних нейронів. У даній системі обробкою даних займається безпосередньо клітина. Сигнал від тіла нейрона розповсюджується по аксонам та дендритам, а потім через синапси (місткі між двома нейронами) до інших нейронів. При цьому, важливо зазначити, що сигнал який передається через синапс може змінюватися

(амплітуда, частота). Обробка даних тілом нейрона в спрощеному баченні представляє із себе суматор вхідних сигналів (від інших нейронів). На основі інтенсивності вхідного сигналу тіло нейрона приймає рішення про передачу сигналу далі, при цьому сигнали, що надходять можуть як сприяти збудженню клітини так і перешкоджати йому. Безпосередньо передача сигналу далі відбується якщо сумарне збудження у тілі нейрона більше за деякий поріг. Звісно, що це дуже спрощене бачення структури біологічного нейрону, проте більшість штучних нейронних мереж послуговується саме цими, простими властивостями [1].

1.1.2 Математичний опис штучних нейронних мереж

Штучна нейронна мережа в найпростішому випадку складається із вхідного шару нейронів, прихованих шарів (їх може бути скільки завгодно або не бути взагалі; їх кількість впливає на складність мережі) та вихідного шару нейронів.

Вхідний шар нейронів по аналогії можна представити у якості сенсорів або датчиків, які приймають якусь інформацію. Приховані шари виконують деякі перетворення із вхідними даними, а вихідний шар нейронів – це по суті результат роботи мережі. Знову ж таки по аналогії у нас є деякий подразник (вхідні дані) і ми хочемо отримати деяку реакцію (вихідні дані). Така аналогія є досить доречною при вирішенні деяких задач на основі нейронних мереж.

Типовою архітектурою нейронної мережі, яку часто зображують при її огляді є повнозв’язна мережа прямого поширення. Це означає, що будь які сигнали від вхідного шару поширюються тільки в одному напрямку (до виході мережі), при цьому кожен нейрон наступного шару пов’язаний із нейронами попереднього і так далі. Структура такої мережі зображена на рисунку 1.2. Якщо один або декілька зв’язків повноз’язної мережі прямого поширення відсутні така мережа називається частково зв’язною (partitially connected).

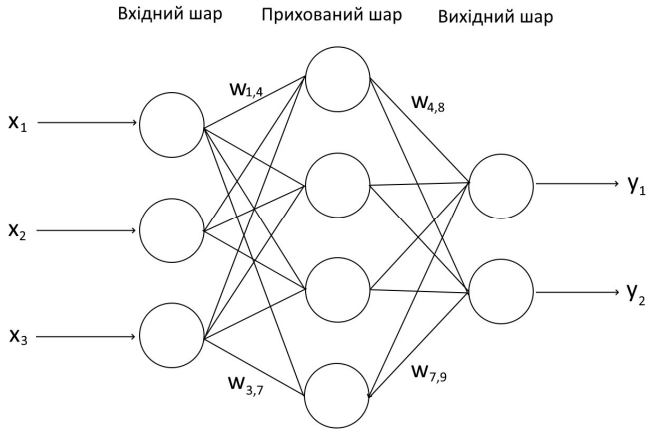


Рисунок 1.2 – Структура повнозв’язної нейронної мережі прямого поширення

Де x – це входи мережі, $w_{i,j}$ – це вага зв’язку і-го та j-го нейронів, y – виходи мережі. Кожен нейрон виконує функцію суматора (по аналогії із тілом клітини у біологічних мережах). Математичний опис визначення вихідного сигналу нейрону представлений у формулі 1.1 [1].

$$h_i = \sigma\left(\sum_{j=1}^{N} V_{ij} x_j\right) \quad (1.1)$$

Значення нейронів попереднього шару $x_{i,j}$ множаться на вагові коефіцієнти wi,j та складаються. Отримане значення називається зваженою сумою. Значення вагових коефіцієнтів для кожного зв’язку різні. Зважена сума є не нормалізованою. Для її нормалізації обирається функція активації σ (sum). Функція активації яка згадується найчастіше при огляді нейронних мереж – це логістична функція (або сигмоїд, формула 1.2 [1]). Також, часто можна зустріти гіперболічний тангенс f(x) = th(x). Мета функції активації – привести вихідні значення нейронів у діапазон, прийнятний для передачі далі по нейронній мережі (як правило це значення від -1 до 1 або від 0 до 1). За аналогією до елекронних систем активаційну функцію можна вважати нелінійним підсилювачем.

$$f(x) = \frac{1}{1 + \exp(-x)} \quad (1.2)$$

Функціональна схема (побудована на основі формули 1.1) будь-якого нейрону повнозв'язної мережі прямого поширення представлена на рисунку 1.3.

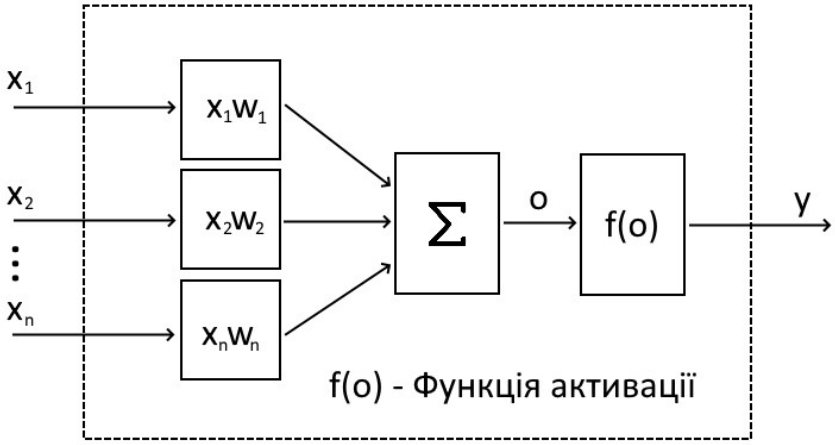


Рисунок 1.3 – Функціональна схема штучного нейрону

Фактично, вся нейронна мережа може бути представлена як дуже складна функція із комплексними входом та виходами. Це надає їй змогу апроксимувати будь-яку іншу функцію із точністю, що визначається її структурою. При цьому входи та виходи мережі можуть бути закодовані будь-якими двійковими даними (текст, кольори пікселів зображення, значення датчиків і т.д і т.п), що надає дуже широкий спектр її застосування [3,4].

1.2. Аналіз стану розвитку нейронних мереж

1.2.1 Основні напрямки дослідження

Дослідження принципів функціонування мозку надихнуло науковців розробити математичну модель яка б повторювала дані принципи. Проте, в ході заглиблення у дане питання виникли інші проблеми які пов'язані із тим, що

мозок, зокрема людини є надзвичайно складною структурою, яку складно описати математично. Проте краще розуміння функціонування нейронів дозволило створити різні моделі для перевірки своїх досліджень. Моделювання, навіть у багатократно спрощеному вигляді дозволила отримати цікаві результати. Властивості таких моделей не тільки частково імітують функції мозку, але й мають інші особливості. На підставі цього подальше дослідження нейронних мереж можна умовно поділити на два напрямки, а саме:

1. Зрозуміти, як все ж таки працює людський мозок на фізичному та психологічному рівні. Зрозуміти свідомість і т.і.
2. На основі зрозумілої моделі розробити нові розрахункові системи (штучні нейронні мережі), що дозволяють вирішувати прикладні задачі, які є надскладні для звичайних алгоритмів.

Власне, розглядаючи штучні нейронні мережі, мова йде саме про другий напрямок дослідження [1].

Штучні нейронні мережі є актуальною темою дослідження приблизно з 1980-х років і до сьогодення. Ми використовуємо лише властивості мозку щодо обробки інформації, спосіб у який він їх проводить і абстрагуємося від мозку вцілому.

1.2.2 Становлення штучних нейронних мереж, як окремої галузі

Вивчення нейронних мереж почалось ще в 1940-х роках. Американський психолог Воррен Маккалох та математик Волтер Піттс запропонували першу в світі модель нейронної мережі М-Р. Вона є дуже простою, але важливою для подальшого розвитку. У моделі алгоритм реалізовано шляхом розгляду нейрона як функціонального логічного пристрою, таким чином, розпочинається теоретичне дослідження моделі штучної нейронної мережі.

На цьому етапі дослідження з’явилося дуже складне питання щодо навчання нейронної мережі. Як було зазначено раніше, нейронну мережу, можна представити як дуже складну функцію. Якщо правильно підібрати її параметри (вагові коефіцієнти для кожного із зв’язків) вона здатна

апроксимувати іншу функцію, навіть дуже складну (це є емпіричним спостереженням, але було виявлено вже пізніше). Як не складно здогадатися – підбирати дані параметрі вручну для невеликих структур складно, а для великих нейронних мереж (сучасні налічують щонайменше тисячі зв'язків) взагалі неможливо. Тому необхідний був алгоритм який дозволить навчати нейронну мережу. Перша модель навчання була запропонована Дональдом Геббом у 1949 році. Він висунув гіпотезу, що процес навчання відбувається на синаптичному інтерфейсі між нейронами, а інтенсивність синаптичних зв'язків змінюється залежно від активності нейронів до і після синапсу. Даний принцип називається правилом Гебба, що використовується й сьогодні при навчанні нейронних мереж у доповненому вигляді.

Пізніше, Розенблат запропонував модель так званого «перцептрона» на основі первинної моделі М-Р (це функціональна схема штучного нейрону, розглянута раніше). В дану модель закладені базові принципі на яких тримається сучасне дослідження штучних нейронних мереж, не залежно від їх складності. Це модель нейронної мережі із безперервним регулюванням вагових коефіцієнтів. Після навчання він може досягти мети класифікації та розпізнавання певного режиму вхідного вектора. Хоча вона відносно проста, це перша справжня нейронна мережа. Розенблатт довів, що двошарові мережі можуть класифікувати вхідні дані, а також запропонував важливий напрямок дослідження для багатошарових мереж із можливістю навчання прихованих (проміжних) шарів мережі для можливості вирішувати більш складні задачі. Однак, навіть така проста за структурою нейронна мережа дозволяла вирішувати задачі передбачення погоди, аналізу електрокардіограм та штучного зору.

Це стало дуже важливим кроком у розвитку штучного інтелекту, адже в ході даного дослідження було доведено, що штучна нейронна мережа, структура якої відповідає в деякій мірі нейрофізіології здатна навчатися і вирішувати задачі [5].

1.2.3 Проблеми штучних нейронних мереж

Після перших досягнень прикладного характеру у навчанні штучних нейронних мереж, вчені деякий час вважали, що для досягнення штучними нейронними мережами інтелекту близького до людського необхідно лише підвищити кількісні характеристики мережі (збільшити кількість нейронів та прихованих шарів мережі). Досить швидко стало зрозуміло, шо цього аж ніяк не достатньо.

Одні із засновників штучного інтелекту Мінський та Пеперт провели математичне дослідження функцій та обмежень перцептрону. Йому не вдалося вирішити проблему класифікації двох типів лінійних нероздільних зразків. Прикладом може слугувати простий лінійний датчик, який не має змоги виконати логічне співвідношення XOR («виключного або»). Дані висновки, переконлива аргументація автора та врахування попередніх розчарувань у штучних нейронних мережах значно сповільнили розвиток штучних нейронних мереж на десятки років.

Повертаючись до сьогодення, важливо зазначити, що більшість із описаних проблем фактично нікуди не зникли, а лише зробили якісний стрибок. Структура нейронної мережі та спосіб її навчання залишаються актуальними проблемами. Алгоритм зворотного поширення помилки використовується дуже часто при навчанні нейронних мереж. Для роботи йому потрібна навчальна вибірка, пара вхід – вихід. На її основі вагові коефіцієнти нейронної мережі корегуються таким чином, щоб отримати прийнятний результат. Проблема полягає у тому, що неможливо впевнено сказати чи зможе мережа навчитися за кінцевий час та яку структуру мережі потрібно обрати для цього. Крім того вже навчена мережа може виявитися не оптимальною адже в основі даного алгоритму лежить метод градієнтного спуску, що може призвести до утворення локальних мінімумів. Інакше кажучи, конфігурація мережі (значення вагових коефіцієнтів) може виявитися не самим оптимальним.

Не зважаючи на великі досягнення у галузі машинного навчання за останні десятиріччя, жодна із штучних нейронних мереж, що використовується сьогодні не є універсальною. Кожна з них має низку недоліків та обмежень які дозволяють їх використовувати тільки для конкретних задач [1,5].

1.3 Особливості функціонування нейронних мереж

1.3.1 Достовірність результату роботи штучного інтелекту

Важливим питанням та характеристикою штучних нейронних мереж є їх надійність. Наприклад, людський мозок в окремих випадках показує неймовірні показники структурною надійності і здібності до самоорганізація та перебудови. Але мова йде не про це, адже в цифровому уявлені нейронна мережа лише потік байт не маючий структурі. Мова йде саме про результат роботи мережі у конкретних задачах.

У роботі майже будь-якої нейронної мережі потрібно розуміти деяку особливість. Для прикладу візьмемо повноз’язну мережу прямого поширення із нелінійною функцією активації, що навчається на основі алгоритму зворотного поширення помилки. Скажем, дана мережа вирішує класичну задачу розпізнавання рукописних цифр. Як було згадано раніше, нейронна мережа може апроксимувати деяку складну функцію якщо правильно підібрати деякі коефіцієнти. Банальною задачею яку часто розглядають як приклад є розпізнавання рукописних цифр. В даному випадку функція на вхід отримує зображення, а на виході ймовірність того що дане зображення є однією з цифр по яким мережа навчається. Але так як ми апроксимуємо оригінальну функцію (фактично її не існує – це набір пар зображення-цифра), то результат буде наближеним. Тобто на виході навчаної нейронної мережі ми ніколи не отримаємо 100% результат по будь-якій із цифр. Насправді це не є проблемою, адже важливо щоб вихідний нейрон, що відповідає за конкретну цифру мав найбільшу ймовірність серед усіх. Проблема полягає в тому, що нейронна мережа може помилятися саме на етапі результату в робочих умовах (коли в неї

немає підказки того, яка конкретна перед нею цифра і їй потрібно видати результат). Нейронні мережі зараз мають дуже низьку ймовірність помилки (98% і більше правильних відповідей). Така низька ймовірність помилки є важливою коли нейронні мережі можуть застосуватися там, де від їх рішення може залежати життя людини.

Проблема насправді може виникнути саме через те, що ми робимо допущення не неспроможність комп'ютера помилятися. Якщо штучна нейронна мережа буде інколи допускати помилки, то ми робимо висновок про її ненадійність. Але у деяких задачах, які раніше вважалися невирішуваними для комп'ютера, нейронні мережі вже показують результати кращі за людину [1].

1.3.2 Основні властивості та переваги

Розглядаючи штучні нейронні мережі та спостерігаючи за результатами їх роботи можна відокремити дві важливі особливості:

1. Розрахунки, які необхідні для функціонування нейронних мереж можуть виконуватися паралельно. Кожен шар нейронної мережі – це набір нейронів для яких необхідно виконати операції зваженої суми та нормалізації, які не залежать від інших нейронів того ж шару.

2. Нейронні мережі здатні самонавчатися та узагальнювати. Узагальнення (generalization) в даному випадку – це здатність давати правильну відповідь на вхідні дані, які не зустрічалися під час навчання. Наприклад, мережа яка вчилася розпізнавати факт наявності на зображенні обличчя повинна відповісти правильно якщо їй надати незнайоме зображення.

Використовуючи дані особливості, нейронні мережі мають змогу вирішувати масштабні задачі, невирішувані, стандартними алгоритмічними засобами. На практиці, як вже було зазначено раніше, універсальних нейронних мереж не існує. Якщо задача занадто складна, доцільно її розбити на декілька простих задач і навчити різні мережі виконувати кожну із них. Або використовувати комбінації нейронних мереж та звичайних алгоритмів.

Важливо розуміти, що нейронні мережі дають лише наближенний результат і стандартні алгоритми можуть виявитися значно ефективнішими у тривіальних задачах. Пошук нових задач де можливе застосування нейронних мереж є окремим дослідженням.

Якщо порівнювати штучні нейронні мережі із стандартними алгоритмами, то їй використання надає наступні корисні властивості системі:

1. Нелінійність (nonlinearity). Штучні нейрони можуть мати як лінійний характер так і нелінійний, що в свою чергу залежить від функції активації. Нейронні мережі, структура яких складається із нелінійних нейронів сама набуває властивість нелінійності. Нелінійність такого виду дещо особлива, адже в такому випадку вона розподілена (distributed) по мережі. Властивість нелінійності є вкрай важливою, коли фізичний механізм, що лежить в основі вхідного сигналу теж є нелінійним (людства мова, зображення, відео тощо). Крім того, алгоритм зворотного поширення помилки, який часто застосовуються при навчанні нейронних мереж використовує модифікований метод стохастичного спуску, для якого необхідно визначити похідну функції активації. Функція активації повинна бути нелінійною на робочому діапазоні.

2. Відображення вихідної інформації через вхідну (input-output mapping). Навчання з вчителем (supervised learning) передбачає наявність деякої вибірки даних, прикладів (training samples). Приклад – це пара із вхідних даних та вихідних, що їм відповідають. Навчання передбачає зміну вагових коефіцієнтів таким чином, щоб на виході мережі отримати достовірний результат. Навчання продовжується до тих пір поки зміни вагових коефіцієнтів стануть незначними.

3. Адаптивність (adaptivity). Нейронним мережам притаманна здібність адаптувати свої вагові коефіцієнти таким чинам щоб відповідати оточенню. Власне, нейронна мережа навчена діяти у конкретній середі може без проблем бути адаптована під роботу у новій середі, якщо її параметри відрізняються не в значній мірі. Крім того, для роботи в нестаціонарній (nonstationary) середі, якій притаманна зміна статистики із часом, можуть бути створені нейронні мережі, які навчаються у реальному часі.

4. Очевидність відповіді (evidential response). На прикладі класичної задачі класифікації за вибіркою можна побудувати нейронну мережі таким чином, щоб при навчанні, на виході вона враховувала не тільки конкретний клас, а й те наскільки мережа упевнена у своїй відповідях, для підвищення достовірності (confidence) рішень, що мережа приймає.

5. Контекстна інформація (contextual information). Знання про об’єкти, що подаються на вхід нейронної мережі зберігаються у вагових коефіцієнтах окремих нейронних зв’язків. При цьому в даній парадигмі усі нейрони пов’язані між собою і жоден довільний нейрон без контексту не несе в собі корисної інформації.

6. Відмовостійкість (fault tolerance). Нейронні мережі, представлені у електронному вигляді потенційно відмовостійкі. Вихід з ладі окремих нейронів або її зв’язків призводить до ускладнення отримання достовірної інформації. Проте, враховуючи спосіб збереження інформації у нейронній мережі (децентралізований та контекстуальний) тільки значні пошкодження структури нейронної мережі можуть призвести до її не можливості виконувати свою роботу.

7. Масштабованість (Scalability). Структура нейронної мережі представлена простими елементами – нейронами та їх зв’язками. Можливості із масштабування нейронних мереж обмежені лише можливостями обчислювальної техніки та адекватністю підходу до поставленої задачі.

8. Одноманітність аналізу та проектування (Uniformity of analysis and design). Нейронні мережі є універсальним інструментом із обробки даних. Одне й те саме проєктне рішення може слугувати для вирішення різних задач. Це обумовлено тим, що базові принципі функціонування різних типів нейронних мереж однакові.

1.4 Класифікація штучних нейронних мереж

Існує надзвичайно багато різних конфігурацій нейронних мереж. Проте, більшість із них можна класифікувати за деякими базовими ознаками. Власне,

нейронні мережі розрізняють за топологіє зв’язків між нейронами та за типом штучних нейронів, що використовується.

Зручним є представлення нейронної мережі у вигляді спрямованого, зваженого графа. В такому уявлені прослідковуються напрямки передачі сигналу у нейронній мережі, а також вага кожного зв’язку.

За способом побудови зв’язків нейронні мережі можна поділити на мережі прямого поширення (у яких граф не матиме петель) та нейронні мережі із зворотними зв’язками [6].

1.4.1 Нейронні мережі прямого поширення

Одношаровий перцептрон. В такій моделі нейронної мережі усі елементи вхідного шару пов’язані із вихідними через зв’язки. Є найпростішою мережею прямого поширення. [7].

Багатошаровий перцептрон Розенблата. Це перцептрон в якому присутній хоча б один прихований (асоціативний) шар. На відміну від одношарового даний тип перцептрона здатний вирішити класичну задачу XOR.

Багатошаровий перцептрон Румельхарта. Даний тип нейронної мережі є частковим випадком перцептрону Розенблата. Відмінністю є те, що навчання відбувається за допомогою методу зворотного поширення помилки. При цьому навчаються усі шари мережі [8].

Згорткові нейронні мережі. Даний тип нейронних мереж надихався принципами закладеними у людській зір. При аналізі зображення ми дивимося на окремі його частини за допомогою деяких фільтрів. Дана операція називається згорткою. Виходом даної мережі я карта ознак зображення яка у задачах класифікації, як правило подається на вхід звичайного багатошарового перцептрону, який вже видає ймовірність належності зображення до окремих класів [9].

Мережа радіальних базисних функцій. Як правило, така мережа включає три шари: вхідний, вихідний та прихований шар нейронів із радіально базисною активаційною функцією. На вхідні нейрони даної нейронної мережі подається

одне й те саме значення. Дана модель мережі дозволяє ефективно виконувати задачі апроксимації функцій [10].

1.4.2 Нейронні мережі із зворотнім зв’язком

Самоорганізаційні карти Кохонена. Особливістю даного типу мереж є навчання без вчителя. Крім того, на відміну від виправлення помилок використовується принцип конкурентного навчання. Тобто, далі передається лише сигнал найбільш схожий на вхідний – інші ігноруються. Даний підхід дозволяє мережі знаходити закономірності у вхідних даних, підсилюючі схожі ознаки [11].

Генеративна змагальна мережа. Прикладом роботи даного типу мереж є генерація людських обличь. Насправді дана система складається із двох нейронних мереж. Одна нейронна мережа намагається згенерувати реалістичне зображення людського обличчя, а інша намагається визначити підробку. Даний тип навчання є навчанням без вчителя [12].

Нейронна мережа Гопфілда. Особливістю даного типу нейронних мереж є наявність симетричної матриці зв’язків. Робота даної мережі зводиться до деякого положення рівноваги (локальний мінімум). Дана структура дозволяє виконувати мережі функцію фільтра та автоасоціативної пам’яті відновлюючи пошкоджені дані на основі запам’ятованих [13].

1.5 Способи застосування нейронних мереж

Штучні нейронні мережі з початку активного періоду їх дослідження знаходять все нові сфери застосування. Властивість нейронних мереж до апроксимації складних функцій дозволяє їй описувати складні нелінійні процеси.

Фізичні процеси. Більшість фізичних процесів є складними. Навіть розрахувати траєкторію польоту тіла із урахуванням багатьох параметрів може бути складно. Але нейронні мережі гарно апроксимують складні функції.

Знаючи деякі параметри атмосфери у різні моменти часу та погодний стан, що їм відповідає можна навчити нейронну мережі передбачати погодні умові на основі нових показників.

Медицина. Здатність нейронних мереж до самоорганізації дозволяє виявляти закономірності між хворобами та пацієнтами (їх показниками). Навчена мережа може діагностувати хворобу пацієнтові на основі попереднього досвіду.

Автоматизовані системи управління. Управління виробництвом, а конкретніше окремими виробничими процесами. Нейронна мережа може навчитися змінювати параметри системи на основі деяких вхідних даних про стан виробництва оптимальним чином.

Класична задача розпізнавання образів або класифікації. Подати на вхід мережі зображення і отримати на виході ймовірнісну оцінку мережі належності цього зображення до того чи іншого класу. Сюди ж входить розпізнавання образів у реальному часі [1,2].

Стискання даних із втратами. Властивість нейронних мереж знаходити спільні ознаки у різних вхідних даних дозволяє використовувати їй для компактного уявлення цих самих даних, наприклад зображень [14].

Висновки до розділу 1

Перший розділ присвячено дослідженню поняття нейронної мережі, аналізу стану розвитку даної галузі. Зокрема було:

- Розглянуто базові принципі, які закладені у роботу нейронних мереж;
- Проаналізовано особливості роботи нейронних мереж;
- Розглянуто переваги та недоліки нейронних мереж у порівнянні із звичайними алгоритмами;
- Проаналізовано стан розвитку нейронних мереж та проблеми з якими зіткалося машинне навчання впродовж свого розвитку;
- Розглянуті основі види нейронних мереж та задачі які вони вирішують.

Резюмуючи даний розділ загальна тенденція розвитку нейронних мереж полягає в тому, що ефективне вирішення кожної окремої задачі потребує створення нового підходу або модифікація існуючого.

2 МЕТОДІ ОБРОБКИ ІНФОРМАЦІЇ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

2.1 Машинний зір. Стиснення графічних образів.

Класифікація зображень та їх компресія тісно пов’язані між собою задачі за використання машинного зору. Класифікація зображень на основі ANN мереж - це задача визначення ймовірності належності зображення до того чи іншого класу. Найбільшу ефективність при вирішенні задач машинна зору дають згорткові нейронні мережі, та їх модифікації. При навчанні такої мережі формуються деякі фільтрі, що застосовуються до вихідного зображення. Як правило, дана операція повторюється багато разів для формування більш складних фільтрів. Даний підхід може бути використаний для стискання в тому числі, адже зображення можна розкласти на деякі складові (ознаки, патерни). На рисунку 2.1.а [15] зображені можливі фільтри для згорткового шару CNN мережі [16].

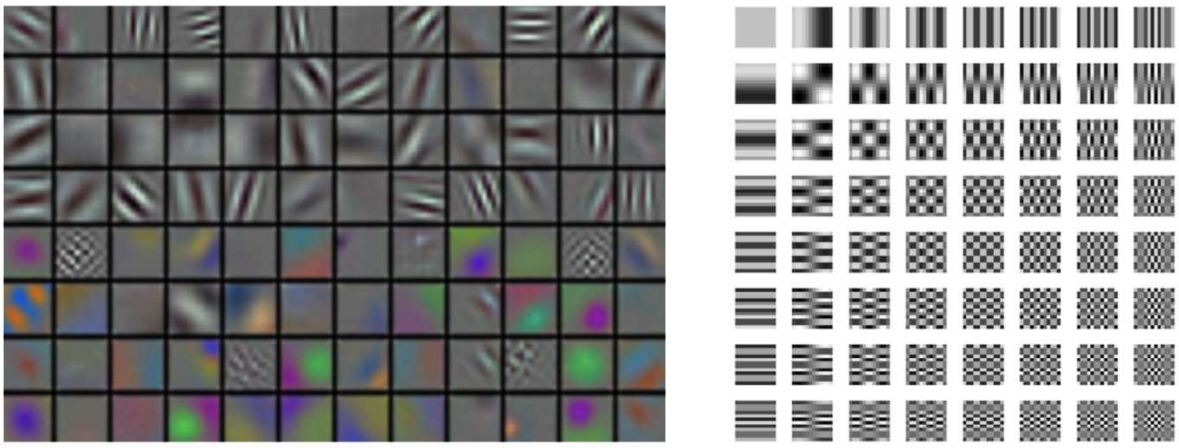


Рисунок 2.1 – Фільтри згорткового шару CNN мережі (а), шаблони DCT (б).

Подібний підхід використовується у дискретному косиносному перетворенні (частина алгоритму стискання зображенні у форматі jpeg). Для DCT визначено 64 шаблони (рисунок 2.1.б [17]), що представляють собою різні

мапи яскравості розміром 8x8. Комбінуючи дані шаблони між собою з деяким коефіцієнтом можна отримати карту яскравості для частини будь-якого зображення. Таким чином для того, щоб отримати карту яскравості деякого зображення достатньо лише знати коефіцієнти для відповідних шаблонів. Компресія відбувається на етапі виключення із кінцевої матриці коефіцієнтів для окремих шаблонів тих складових, що відповідають більш високим частотам. Таким чином, обсяг інформації до зберігання зменшується за рахунок виключення тієї інформації, що менше впливає на кінцеве зображення [17].

Проте, у нейронних мереж в даному підході є значна перевага, яка полягає в тому, що ми можемо формувати ознаки більш високого рівня для різних класів зображень.

2.1.1 Bottleneck архітектура

Процес навчання нейронної мережі це знаходження невідомих параметрів функції для того щоб на деякий вхід X вона давала вихід Y (апроксимація). Інакше кажучи, навчання – це зменшення помилки між виходом мережі та очікуваними даними. Зараз, як правило для вирішення складних задач використовується глибоке машинне навчання. Даний термін застосовується до нейронних мереж з великою кількістю внутрішніх шарів (насправді, від двох та більше). Що відбувається на прихованих шарах мережі здебільшого невідомо. Як було зазначено раніше, будь-який нейрон без контексту не несе в собі ніякої інформації. Тому таку модель називають чорним ящиком.

Архітектура Bottleneck в загальному випадку не визначає структуру мережі цілковито. Натомість, вона передбачає наявність деякого шару нейронної мережі який буде «вузьким» її місцем. Тобто, кількість нейронів цього шару (або фільтрів у випадку із CNN, що майже те саме) буде менша порівняно із іншими. На перший погляд це мало корисно, адже згідно із концепцією нерівності обробки даних (Data Processing Inequality, DPI) – жодна обробка даних не може збільшити їх ентропію. Інакше кажучи обробка даних не може

збільшити кількість інформації, крім того даний шар мережі очевидно зменшить кількість інформації на його виході порівняно із входом. Проте, саме такий підхід може бути корисним у вирішенні задач компресії. Важливо розуміти, що коли мова йде про нейронні мережі – це завжди приблизний результат, а отже і стиснення із втратами. Є винятки, коли нейронні мережі дозволяють стискати дані без втрат, проте й принцип роботи таких мереж значно відрізняється. На рисунку 2.2 зображена модель bottleneck мережі.

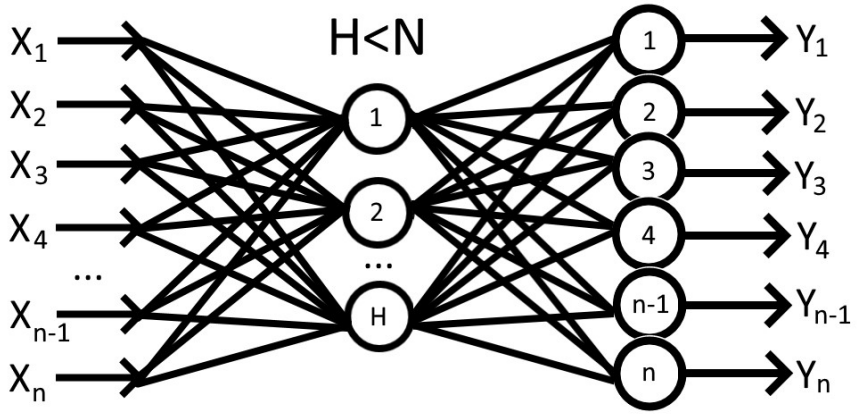


Рисунок 2.2 – Приклад bottleneck архітектури нейронної мережі.

Для здійснення стискання даних за допомогою такої мережі необхідно в першу чергу її цьому навчити. На вхід мережі подаєм деяке зображення (насправді, це можуть бути будь-які дані для яких стискання із втратами є прийнятним: відео, звук тощо) і на виході очікуєм те саме зображення. За рахунок того, що деякий прихований шар мережі має розмірність менший за вхідний (по суті розмір зображення), мережа під час навчання повинна навчитися репрезентувати вхідні дані, але в стислому вигляді. Тоді ми можете розділити мережу, що навчилася на дві: кодер та декодер. При цьому передавати каналом зв’язку слід вихідні значення bottleneck шару мережі. У прикладі bottleneck даної архітектури мережі зображені повнозв’язні шари. Насправді, дана архітектура не використовується у реальних задачах, проте основна ідея того, що дані за допомогою нейронної мережі можна репрезентувати у стислому вигляді зберігається у більш складних підходах [18,19].

2.1.2 Переваги CNN мереж у задачах комп’ютерного зору

При розв’язанні задач комп’ютерного зору зазвичай використовують згорткові нейронні мережі. У порівнянні із повнов’язними мережами вони мають менше параметрів, які необхідно корегувати під час навчання. Крім того, вони набагато краще аналізують зображення, за рахунок застосування до окремих частин зображення одних і тих самих нейронів (нейронів фільтру).

Рисунок 2.3 – Приклад навчальних даних для розпізнавання цифр

Якщо повнозв’язна мережа навчилася розрізняти рукописні цифри, які були відцентровані на вхідному зображенні (рис. 2.3.а,б), то в неї виникнуть проблеми якщо при тестуванні мережі на вхід подати зображення цифра на якому не відцентрова, або їх взагалі декілька (рис. 2.3.в), то мережа дасть невірну відповідь. Це пов’язано із тим, що на вхідні пікселі зображення реагують лише конкретні нейрони. Власне, цього недоліку позбавлені згорткові нейронні мережі, адже замість цього, зображення аналізується лише декількома групами нейронів, що називаються фільтрами. При цьому кожен фільтр аналізує усе зображення формуючи вихід. Даний принцип дотичний до того як влаштований людський зір. Оптичні нерви з’єднуються не з усіма нейронами зорової кори, а лише з обмеженою їх кількістю [16].

2.2 Принцип роботи згорткових нейронних мереж

Згорткові нейронні мережі оперують такими поняттями як тензор, мапа ознак, згортковий шар, шар субдискретизації (також зустрічається пулінг, семплінг).

Скаляр, вектор та матриця є частковими випадками тензора (0,1 та 2 вимірного відповідно). При роботі із CNN мережами зустрічаються тензори вищого порядку [20]. Мапа ознак представляє із себе тензор, що утворений після операції згортки. Шар субдискретизації призначений для змінення розмірності тензору та виділення домінуючих значень у мапі ознак.

2.2.1. Операція згортки

Основними елементами операції згортки є вхідний тензор (мапа ознак або зображення) та згорткові фільтрі. Результатом операції згортки є мапи ознак у кількості рівній кількості згорткових фільтрів. Розмір згорткового фільтру зазвичай обирається 3 на 3, при цьому глибина фільтру відповідає кількості каналів вхідного тензору. Якщо на вхід згорткового шару ми подаємо звичайне зображення, то аналізувати його потрібно фільтром розміром k_1xk_2x3, де 3 – кількість каналів зображення (R,G,B).

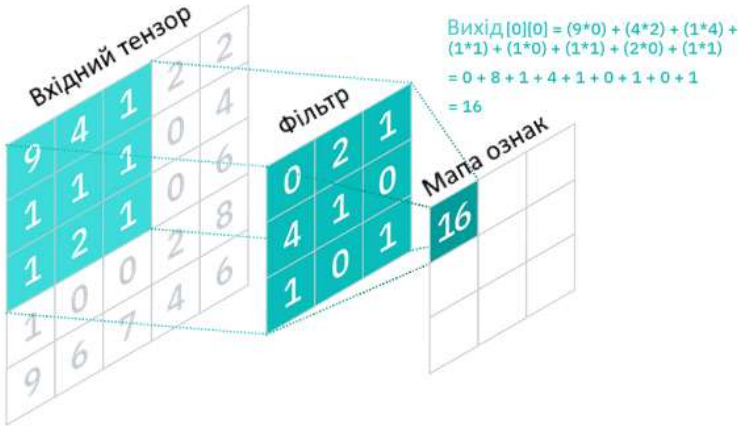


Рисунок 2.4 – Візуалізація операції згортки

Також, існує такий параметр, як stride (крок фільтру). Чим більший крок

фільтру, тим більшу частину вихідного зображення буде охоплено на кожному наступному кроці, і тим менше буде розмірність вихідного тензору (мапи ознак).

Якщо на вхід ми подаємо n мап ознак, то потрібен фільтр розмірністю $k_1 x k_2 x n$. Фільтр рухається по зображенню, і елементи тензору фільтра множаться на відповідні елементи вхідного тензору, після чого складаються. Процес розрахунку елементу мапи ознак зображений на рисунку 2.4 [21]. До даної суми може додаватися bias (зсув). Даний параметр окремий для кожного фільтру та навчається. Отримана сума є активацією елемента мапи ознак.

Отримані значення активації необхідно подати на вхід функції активації. Виходи функції активації формують вихід згорткового шару (мапи ознак).

Основний критерій вибору функції активації – це її емпірично досліджена ефективність. До базових критеріїв належить її нелінійність (адже за допомогою лінійної функції можна описати лише лінійну функцію) та диференційованість. Наразі, для CNN мереж використовується переважно ReLU (рисунок 2.5) Іноді, від'ємну частину графіка множать на деякий коефіцієнт (Leaked ReLU), також цей коефіцієнт може бути змінним параметром (Parametrical ReLU) [22].

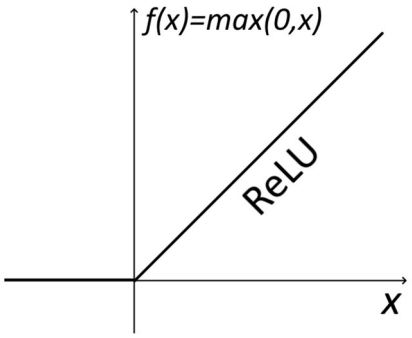


Рисунок 2.5 – Функція активації ReLU

Одна із переваг ReLU – це її простота з точки зору кількості математичних операцій необхідних на її обчислення.

Проходження фільтром по зображенню (або мапі ознак) зменшує його

розмірність. Це не завжди зручно і для того, щоб цього уникнути використовують zero padding (доповнення нулями, рисунок 2.6 [23]). Також, дана операція може бути використана для збільшення розмірності виходу порівняно із входом [23].

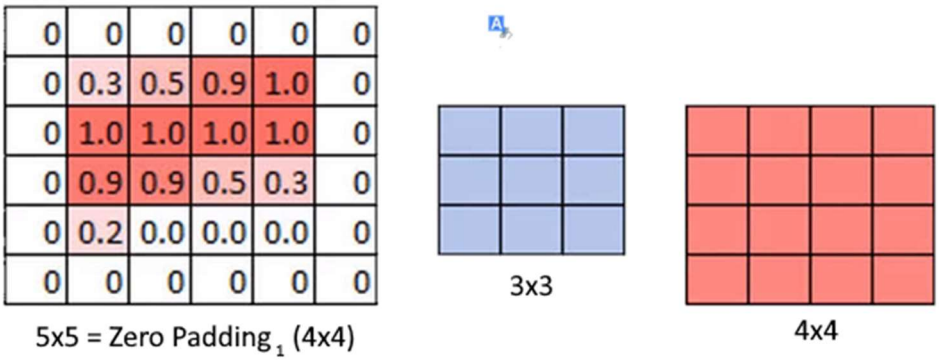


Рисунок 2.6 – Доповнення вхідного тензору нулями

Кількість параметрів в згортковому шарі досить мала, порівняно із повнозв'язним. Це (k_1 x k_2 + 1) x n, де n – кількість фільтрів. При цьому на відміну від повнозв'язною мережі кількість параметрів не залежить від розміру зображення.

2.2.2. Субдискретизація

Дана операція необхідна для зменшення або збільшення розмірності вихідного тензору (але не його глибини).

Для зменшення розмірності вихідного тензору використовують Max Pooling.

Як і у випадку із згорткою у нас є такі параметри як розмір фільтру та крок. Параметрів даний шар не має, тому фільтр в даному випадку – це вікно деякого розміру яким ми проходимо по зображенню. Ми обираємо найбільше значення із кожного вікна і записуємо його у вихідний тензор. Таким чином, ми не тільки зменшуємо складність розрахунку, а й виокремлюємо домінантні ознаки із вихідної мапи. Це відбувається за рахунок того, що саме найбільші значення

відповідають деякій ознаці. Операції max pooling зображена на рисунку 2.7 [24].

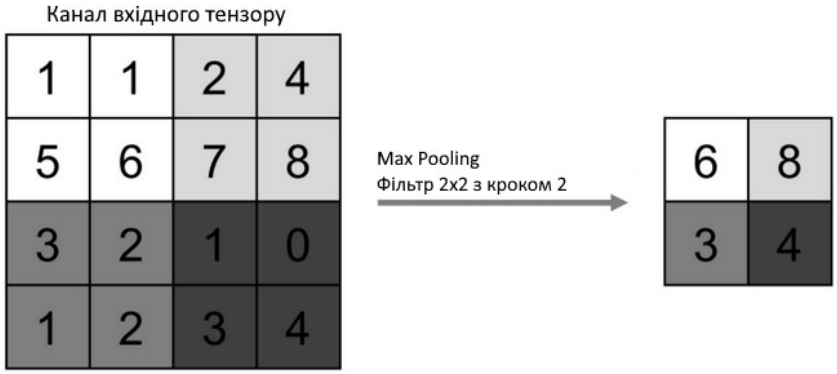


Рисунок 2.7 – Операція Max Pooling

Як правило, згортковий шар, що слідує після даного має кількість фільтрів пропорційно більшу порівняно із попереднім. Це дозволяє зберегти складність розрахунку кожного наступного шару на тому ж рівні, крім того – це дозволяє отримувати більше ознак високого рівня.

Для збільшення розмірів вихідного тензору використовується операція Up Sampling. Вона може знадобитися у задачах стиснення зображень. Так як виходом мережі є зображення тієї ж розмірності, дана операція дозволить повернути початкову розмірність зображення.

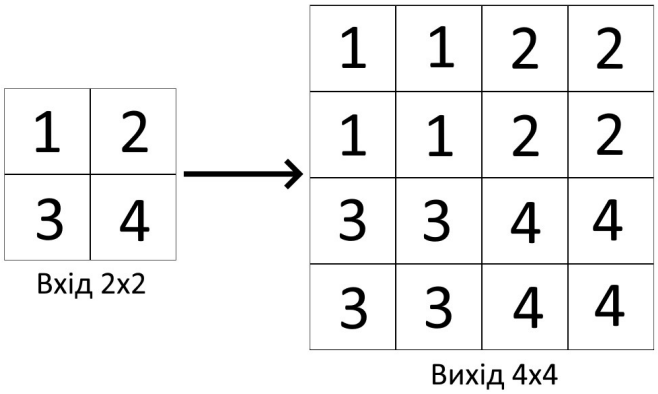


Рисунок 2.8 – Операція Up-Sampling

Так як розмірність вихідного тензору буде пропорційно більше, то потрібно чимось доповнити інформацію, якої не вистачає. Як правило, пусті елементи вихідного тензору заповнюється із вхідного як це зображено на

рисунку 2.8 [25].

2.3. Навчання згорткових нейронних мереж за допомогою алгоритму Back Propagation

2.3.1. Функції втрат у задачах класифікації та регресії

Функція втрат є важливим компонентом навчання нейронних мереж. Дана функція представляє із себе оцінку того наскільки мережа помилилася у своїй відповіді. Правильно побудована функція втрат – це основа того, що мережа навчиться прогнозувати правильні відповіді на тестових даних. Відповіддю мережі як правило є скалярна величина для задач регресії (координати деякої ознаки на зображенні, колір пікселя т.і), або ймовірність належності до деякого класу у задачах класифікації [27].

Задача класифікації. Кожна із зазначених типів мереж має свої особливості. Як правило, згорткова нейронна мережа у задачах класифікації закінчується повнозв'язною мережею (рисунок 2.9 [28]).

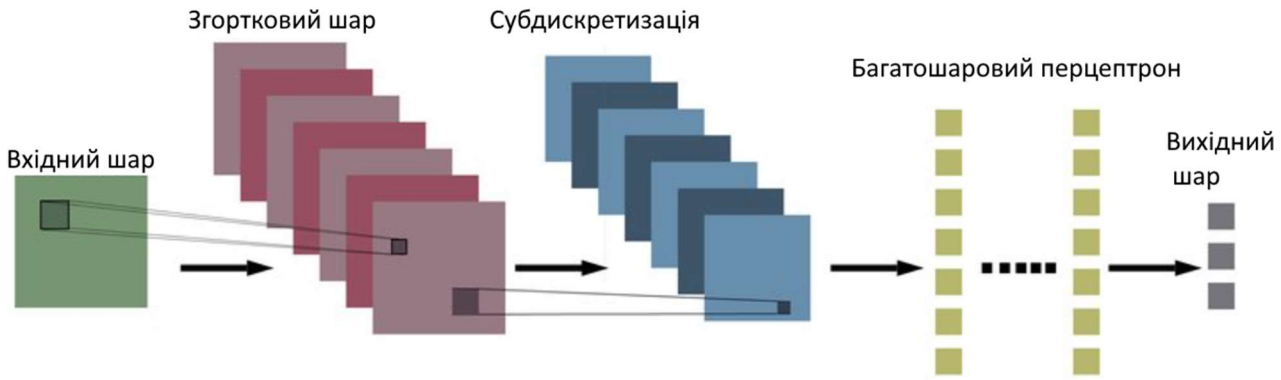


Рисунок 2.9 – Приклад CNN мережі класифікації

Кількість нейронів останнього шару повнозв'язної частини мережі відповідає кількості класів які розрізняє дана мережа. При цьому, бажано щоб значення нейронів вихідного шару були в діапазоні від 0 до 1 (ймовірність належності до деякого класу), а сума цих значень була рівна 1 (адже сумарна

ймовірність повинна бути 100%). Для досягнення такого результату використовується функція SoftMax, що має вигляд (формула 2.1):

$$SoftMax(X) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (2.1)$$

В якості функції втрат як правило використовується Cross Entropy, що має вигляд (формула 2.2):

$$CE(Y,T) = -\sum_i T_i \cdot \ln(Y_i) \quad (2.2)$$

, де T – вектор правильних відповідей, а Y – вихідний вектор останнього шару нейронної мережі. Як правило останній шар має функцію активації Soft Max або іншу функцію нормалізація (для бінарної класифікації Sigmoid, а також деякі інші).

Задача регресії. Задачі регресії у машинному навчанні відповідають за навчання мережі, що вміє передбачати скалярні значення. Можна сказали, що задача класифікації – це частковий випадок задачі регресії, адже ймовірність (або номер) правильного класу теж є скаляром, проте ці два типи задач досить сильно відрізняються при побудові мережі і тому будь-які задачі у машинному навчанні як правило зводяться до задач класифікації або регресії (або їх комбінацій на різному рівні).

Розповсюдженою функцією втрат для задач регресії є середньоквадратичне відхилення (MSE, Mean Squared Error). Дана функція представляє із себе середнє між квадратами різниці елементів вектору вихідних значень останнього шару мережі (прогнозовані значення) та елементами вектору із правильними відповідями (формула 2.3)[29].

$$MSE(Y,T) = \frac{1}{N}\sum_{i=0}^{N}(Y_i - T_i)^2 \quad (2.3)$$

Як видно із формули, через квадрат MSE чутливий до великих значень дисперсії у відповіді. Якщо маємо кілька спостережень із приблизно однаковим вектором ознак, оптимальним прогнозом буде їх середнє цільове значення. Таким чином, дана функція втрат оптимальна у тих випадках коли вектори спостережень мають нормальний розподіл та відмінність у правильних відповідях та відповідях мережі повинна накладати високий штраф (квадрат різниці).

Недоліком MSE є те, що ми аналізуємо дані по одному елементу. Наприклад, для зображення такий підхід є не дуже вдалим. По перше квадратична помилка у випадку із зображенням не потрібна, адже відмінність у кольорі пікселя може бути досить великою, але фактично для прийняття зображення буде нормальним. Крім того в середньому значення кольору на виході мережі можуть мати високий рівень збіжності із правильною відповіддю, але візуально зображення буде спотворено.

Таким чином, виникає потреба аналізувати окремі виходи мережі (колір пікселя) із урахуванням сусідніх значень. Для вирішення даної проблеми може бути задіяний індекс структурної подібності (SSIM, Structure Similarity, формула 2.4).

Його особливість полягає у тому, що заміть порівняння значень окремих пікселів ми порівнюємо значення пікселів групами (у деякому вікні). Зазвичай розмір даного вікна обирається невеликий (наприклад 8x8), адже в противному випадку кількість додаткових розрахунків буде зростати в геометричній прогресії, що буде особливо помітно при достатньо великих розмірах зображень.

$$\mathrm{SSIM}(p) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \cdot \frac{2\sigma_{xy} + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \quad (2.4)$$

, де $C_{1,2}$ – константи, $\mu_{x,y}$ – середнє входу та виходу, $\sigma_{x,y}$ – дисперсія входу та виходу.

2.3.2 Метод градієнтного спуску

Навчання мережі – це процес оптимізації функції помилки. Як правило – це навчання із вчителем за допомогою промаркованих даних (спостережень), що називаються датасетом. У машинному навчанні основою для всіх алгоритмів оптимізації функції втрат є стохастичний градієнтний спуск, що засновуються на ланцюговому правилі диференціювання. Приклад застосування даного правила до визначення похідних для вхідних елементів останнього шару нейронної мережі (формула 2.5):

$$\frac{dE}{dx} = \frac{dE}{dy}\frac{dy}{dx} \quad (2.5)$$

В даному випадку dE/dx – це похідна вхідного вектору по функції втрат; dE/dy – похідна вихідного вектору по функції втрат; dy/dx – похідна вхідного вектору відносно вихідного. Дане правило застосовується для будь-якого шару мережі. Звісно, усі функції представлені у мережі повинні мати похідну у робочому діапазоні не враховуючи деякі виключення (для функції ReLU(x) = max(0,x) похідна у точці 0 невизначена, проте її беруть рівною одиниці. На розрахунки це ніяк не впливає) [30,31].

Розрізняють декілька різновидів градієнтного спуску, які відрізняються кількістю спостережень (даних із виборки) які беруть участь в одному кроці градієнта:

1. Класичний градієнтний спуск. В даному випадку градієнт для деякого

параметра мережі буде середнім арифметичним через градієнти для кожного із спостережень. Якщо спостережень дуже багато, то продуктивність роботи такого підходу буде дуже низькою.

2. Стохастичний градієнтний спуск (SGD, Stochastic Gradient Descent). Градієнт робить крок для кожного спостереження. Інша крійність, недоліком якої є те, що розраховуючи градієнт для кожного окремого спостеження ми маємо високий рівень шуму при навчанні мережі (якщо нормальний розподіл вхідних даних має досить велику дисперсію, що зустрічається досить часто).
3. Mini-Batch SGD. Градієнт усереднюється для деякої кількості спостережень. Даний підхід є найбільш оптимальним. Розмір міні-батчу (пакету) є гіперпараметром [32].

2.3.3 Розповсюдження градієнту мережею у зворотному напрямку

Градієнт функції втрат взагалом буде залежати безпосередньо від функції втрат та функкції активації останнього шару мережі.

Проте якщо використовувати Soft Max + Cross Entropy або MSE та лінійну функцію активації (тобто відсутність функції активації f(x) = x) градієнт функції втрат буде рівний (формула 2.6).

$$\nabla_y E = \bar{y} - \bar{t} \tag{2.6}$$

, де y – вектор вихідних значень, t – вектор цільових значень (правильних відповідей для спостереження) [33].

Розповсюдження градієнту через повнозв'язні шари. Розрахунки у середині нейронної мережі зручно представляти у вигляді послідовних тензорних операцій (операцій над векторамми, матрицями та тензорами більшої розмірності) як при прямому поширенні сигналу (Feed Forward) так і при зворотному (Back Propagation).

Ітерація навчання мережі в будь-якому випадку починається із прямого поширення сигналу через мережу для того, щоб зробити оцінку помилки за даних параметрів. Для розрахунку градієнту необхідно визначити часткову похідну відносно функції втрат послуговуючись правилом диференціювання складної функції (адже функція втрат є функцією від функцію, що в свою чергу є функцією іншої складної функції.. і т.д до входу мережі).

Виходи повнозв'язного шару при прямому поширенні сигналу визначаються за формулою 2.7:

$$y = f(W \cdot x + b) \tag{2.7}$$

, де **y** – вектор вихідних значень; **W** – матриця вагових коефіцієнтів; **x** – вектор вхідних значень; **b** – зсуви; **f** – функція активації (нелінійність).

Вираз **a = W·x+b** називається активацією, а **W·x** – зваженою сумою.

Таким чином, необхідно визначити градієнт вхідного вектору по функцію втрат, для того, щоб передати його далі (в сторону вхідного шару нейронної мережі) та градієнт для параметрів, що навчаються (зсуви та вагові коефіцієнти), для того щоб скорегувати їх значення в зворотному від градієнту напрямку (зменшити функцію помилки відносно даного параметру, а отже і для моделі в цілому). Градієнт вихідних значень від функції втрат відомий $\nabla_y E$ (він може бути отриманий із функції втрат безпосередньо якщо це останній шар або із іншого шару мережі для його даний градієнт буде градієнтом вхідних значень відносно функції втрат).

В першу чергу необхідно визначити градієнт активації відносно функції втрат. Це є добутком похідної функції активації в точці **a** та значенням вихідного градієнту $\nabla_y E$ (формула 2.8).

$$\nabla_a E = f'(a) \cdot \nabla_y E \tag{2.8}$$

Як правило для прихованих шарів нейронної мережі використовується

функція активації ReLU(x) = max(0,x). Похідна, як і сама функція досить проста (формула 2.9):

$$ReLU'(x) = \begin{cases} 1 \; if \; x \geq 0 \\ 0 \; if \; x < 0 \end{cases} \tag{2.9}$$

Градієнт для зсувів **b** дорівнює градієнту активації (формула 2.10). Це випливає із формули 2.7. Адже похідна вектора зсуву через активацію дорівнює одиниці.

$$\nabla_b E = \nabla_a E \tag{2.10}$$

Наступним кроком є визначення градієнту для матриці вагових коефіцієнтів $\nabla_W E$ та для вхідного вектору $\nabla_x E$. Для визначення градієнту для матриці вагових коефіцієнтів необхідно визначити її похідну відносно активації. Як видно із формули 2.7 вона буде рівна x, вектору вхідних значень (формула 2.11):

$$\nabla_W E = \nabla_a E \cdot x \tag{2.11}$$

В свою чергу, похідна для вектору вхідних значень відносно активації буде дорівнювати вектору w (елементи матриці W, що пов'язують вхідні значення та відповідну активацію). Таким чином, маємо (формула 2.12):

$$\nabla_x E = \nabla_a E \cdot W^T \tag{2.12}$$

При цьому, якщо шар нейронної мережі йде першим після вхідного, то розрахунок градієнту для вхідних значень не потрібен [34, 35].

Розповсюдження градієнту через згорткові шари. Пряме розповсюдження через згортковий шар відбувається за наступною формулою

2.13:

$$Y = f(x * W + b) \tag{2.13}$$

, де * - це операція згортки, що була представлена раніше.

Важливо, що в даному випадку Y та x – це тензори третього порядку. Вони мають висоту, ширину, а також глибину (для x – це кількість вхідних каналів, а для Y - це кількість згорткових фільтрів). W є тензором 4-го порядку. Він має ширину висоту (розміри ядра згортки), а також номери вхідного каналу та фільтра.

В першу чергу необхідно визначити градієнт активації a = x*W+b. Так як вихідний градієнт нам відомий, необхідно знайти похідну активації відносно виході мережі. Як і у випадку із повнозв'язним шаром це буде похідна функції активації. Таким чином, градієнт активації визначається за формулою 2.14:

$$\nabla_a E = f'(a) \cdot \nabla_y E \tag{2.14}$$

Відмінність полягає у тому, що операції проводяться над 3-вимірним тензором, а не вектором.

Знаючи градієнт для активації можна визначити градієнт для зсувів. У випадку із згортковим шаром ми маємо лише один зсув для кожного каналу фільтра (позначемо номер фільтра як **n**). Тобто зсув чинить вплив на усі нейрони вихідного шару n-го каналу. Для того, щоб знайти градієнт зсуву, треба знайти сума значень вихідного градієнту для n-го каналу фільтру (формула 2.15):

$$\nabla_{b(n)} E = \sum \nabla_{a(n)} E \tag{2.15}$$

Також, знаючи градієнт активації можна визначити градієнт для вагових

коефіцієнтів згорткових фільтрів. Згорткові фільтри мають *c* окремих каналів для кожного каналу вхідного тензора, а також n – каналів для кожного каналу вихідного тензора. Таким чином, за формулою 2.13 у розрахунку n-го каналу активації приймали участь усі канали вхідного тензору (адже їх зважена сума поелементно складалася під час згортки), а також усі канали фільтру згортки (формула 2.16).

$$\nabla_{W(n,c)}E = \begin{pmatrix} \frac{dE}{da_{2,2}}x_{1,1} & \frac{dE}{da_{2,1}}x_{1,1} + \frac{dE}{da_{2,2}}x_{1,2} & \frac{dE}{da_{2,1}}x_{1,2} \\ \frac{dE}{da_{1,2}}x_{1,1} + \frac{dE}{da_{2,2}}x_{2,1} & \frac{dE}{da_{1,1}}x_{1,1} + \frac{dE}{da_{1,2}}x_{1,2} + \frac{dE}{da_{2,1}}x_{2,1} + \frac{dE}{da_{2,2}}x_{2,2} & \frac{dE}{da_{1,2}}x_{12} + \frac{dE}{da_{2,1}}x_{2,2} \\ \frac{dE}{da_{1,2}}x_{2,1} & \frac{dE}{da_{1,1}}x_{2,1} + \frac{dE}{da_{1,2}}x_{22} & \frac{dE}{da_{1,2}}x_{2,2} \end{pmatrix} \quad (2.16)$$

Насправді, знаходження градієнту для вагових коефіцієнтів можна преставити як операцію згортки вхідного тензора із градієнтом активації в якості фільтра (формула 2.17):

$$\nabla_{W(n,c)}E = x_c * \nabla_{a(n)}E \quad (2.17)$$

Дана операція передбачає, що до вхідного тензору був застосований zero padding на етапі згортки (same padding) відповідно до розмірів фільтра.

Останнім етапом розрахунку градієнту згорткового шару є розрахунок градієнту вхідних значень.

Значення вхідного тензору пов’язані із активацією через вагові коефіцієнти. При розрахунку n-того каналу карти активації використовуються усі канали с згорткового фільтру n для аналізу вхідного тензору (формула 2.18):

$$\nabla_{x(c)}E = \begin{pmatrix} \frac{dE}{da_{11}}w_{22} + \frac{dE}{da_{12}}w_{21} + & \frac{dE}{da_{11}}w_{23} + \frac{dE}{da_{12}}w_{22} + \\ \frac{dE}{da_{21}}w_{12} + \frac{dE}{da_{22}}w_{11} & \frac{dE}{da_{21}}w_{13} + \frac{dE}{da_{22}}w_{12} \\ \frac{dE}{da_{11}}w_{32} + \frac{dE}{da_{12}}w_{31} + & \frac{dE}{da_{11}}w_{33} + \frac{dE}{da_{12}}w_{32} + \\ \frac{dE}{da_{21}}w_{22} + \frac{dE}{da_{22}}w_{21} & \frac{dE}{da_{21}}w_{23} + \frac{dE}{da_{22}}w_{22} \end{pmatrix} \quad (2.18)$$

Формулу 2.17 можна інтерпретувати як операцію згортки градієнтів активації де в якості фільтра вагові коефіцієнти згорткового фільтру, але з тим зауваженням, що порядок слідування індексів ширини та висоти інвертований (формула 2.19).

$$W = \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{pmatrix} \quad \widetilde{W} = 180^{\circ}(W) = \begin{pmatrix} w_{3,3} & w_{3,2} & w_{3,1} \\ w_{2,3} & w_{2,2} & w_{2,1} \\ w_{1,3} & w_{1,2} & w_{1,1} \end{pmatrix} \quad (2.19)$$

Крім того для збереження розмірності необхідно до тензору градієнтів активації застосувати операцію zero padding. Остаточно формула для визначення вхідних градієнтів має вигляд (формула 2.20) [36, 37]:

$$W\nabla_{x(c)}E = \sum ZeroPadding(\nabla_a E) * \widetilde{W} \quad (2.20)$$

Як і у випадку із повнозв'язним шаром, якщо згортковий шар йде одразу після вхідного (в задачах комп'ютерного зору, власне так і є) то розраховувати вхідний градієнт немає сенсу.

Взагалом, представлені розрахунки є досить простими якщо розглядати їх як набір послідовних тензорних операцій.

Розповсюдження градієнту через Max Pooling шари. Як і у випадку з іншими шарами нейронної мережі для визначення градієнту шару максимального об'єднання (також Max Pooling, Down Sampling) необхідно

проаналізувати яким чином сигнал розповсюджується через даний шар в прямому напрямку. Для цього представимо операцію вибору максимального як деякий ваговий коефіцієнт, що не навчається (формула 2.21):

$$w_i = \begin{cases} 1 & if\ x_i = max(x_1, x_2, .., x_n) \\ 0 & otherwise \end{cases} \tag{2.21}$$

Тоді саму операцію згортки можна представити, як добуток вхідних значень на дані вагові коефіцієнти (формула 2.22):

$$y = \sum_{i=0}^{n} w_i x_i \tag{2.22}$$

Похідна для вхідного шару відносно активації (операції максимального об'єднаня) – це значення «вагового коефіцієнту» для конкретного елементу (формула 2.23).

$$\frac{dy}{dx_i} = w_i = \begin{cases} 1 & if x_i = max(x_1, x_2, .., x_n) \\ 0 & otherwise \end{cases} \tag{2.23}$$

Похідна для вхідного шару відсносно виході мережі – це добуток градієнту активації на градієнт виходу. Таким чином, градієнт для нейронів входу можна визначити за наступними співвідношенням (формула 2.24)

$$\frac{dE}{dx_i} = \frac{dE}{dy_i}\frac{dy}{dx_i} = \frac{dE}{dy_i} w_i \tag{2.24}$$

Інакше кажучи, градієнт розповсюджується лише через той елемент вхідного тензору який після операції максимального об’єднання був переданий дані мережею. Візуалізації даної операції представлена на рисунку 2.10.

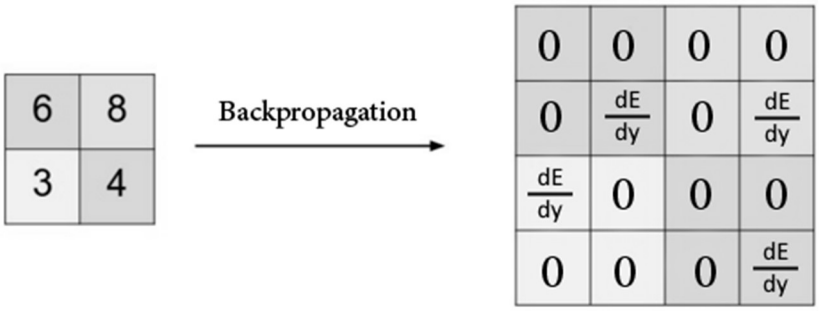


Рисунок 2.10 – Візуалізація Back Propagation через Max Pooling шар мережі

На рисунку dE/dy – це похідна виходу для групи елементів вхідного тензору. Розмір групи визначається розміром фільтра та кроком (в даному випадку 2x2 із кроком 2).

2.3.4 Методи оптимізації функції втрат на основі градієнтного спуску

Під час огляду алгоритму Back Propagation було зазначено, що він працює на основі градієнтного спуску, який в свою чергу має декілька різновидів, відмінність яких полягає у кількості спостережень, які беруть участь в одному кроці градієнту для деякого параметру мережу. Проте, загальне співвідношення для оновлення параметрів залишається однаковим (формула 2.25):

$$param_{new} = param_{old} - \alpha \cdot \nabla_{param} E \tag{2.25}$$

, де α – швидкість навчання; $\nabla_{param} E$ – значення градієнту параметру відносно функції втрат (для одного спостереження, або середнє арифметичне для групи спостережень).

Як показує практика такий підхід може бути не завжди ефективним. Однією з очевидних проблем є ділянки, де функція втрат для даного параметру все ще досить велика, проте похідна в конкретній точці дуже мала (локальні мінімуму). В такому випадку крок в напрямку антиградієнта буде занадто малий і навчання може застрягти в одному мінімумі. Частково дана проблем вирішується більш високою швидкістю навчання, проте якщо збільшити її

занадто помилка може не зійтися зовсім (значення параметру не зможе стабілізуватися в одному мінімумі).

Одним із ефективним способів вирішення даної проблеми є застосування імпульсу (momentum). Ідея полягає в тому, щоб враховувати значення попередніх градієнтів для того, щоб імітувати інерцію. На практиці вираховується експоненційне ковзне середнє значення градієнту, адже зберігати в пам'яті всю історію градієнту та проводити над нею математичні операції було б вкрай неефективним рішенням (формули 2.26, 2.27):

$$\mu_t = \beta\mu_{t-1} + (1-\beta)\nabla_{param}E \tag{2.26}$$

$$param_{new} = param_{old} - \alpha \cdot \mu_t \tag{2.27}$$

, де β – коефіцієнт моменту (як правило беруть рівним 0.9).

Недоліком усіх модифікацій градієнтного спуску та momentum є те, що швидкість навчання залишається постійною. Насправді, зменшувати швидкість навчання з часом є корисним, адже це дозволяє знаходити кращі локальні мінімуми функції які залишаються недосяжними через велику швидкість. Дану проблему намагається вирішити AdaGrad (формули 2.28-2.30):

$$\tilde{\alpha} = \frac{\alpha}{1 + (t-1)\mu} \tag{2.28}$$

$$sum_t = sum_{t-1} + \nabla_{param}E^2 \tag{2.29}$$

$$param_{new} = param_{old} - \tilde{\alpha}\frac{\nabla_{param}E}{\sqrt{sum_t} + \varepsilon} \tag{2.30}$$

, де ε – мала константа для запобігання ділення на 0.

Даний оптимізатор змінює швидкість навчання α для кожного параметра та на кожному кроці «t». Перевага AdaGrad полягає в тому, що він усуває необхідність налаштування швидкості навчання.

Його основним недоліком є накопичення квадратів градієнтів ($\nabla_{param}E$) у

знаменнику. Оскільки кожен такий квадрат завжди є додатним, накопичена сума продовжує зростати під час навчання, спричиняючи скорочення швидкості навчання та стаючи нескінченно малою, що надалі призводить до проблеми зникнення градієнта через малі значення його оновлення.

Дану проблему намагається вирішити оптимізатор AdaDelta. Співвідношення для даного оптимізатор мають вигляд (формули 2.31-2.34):

$$sum_t = sum_{t-1} + \nabla_{param} E^2 \tag{2.31}$$

$$v_t = sum_{t-1}\rho + (1 - \rho) \cdot \nabla_{param} E^2 \tag{2.32}$$

$$\Delta x_t = \frac{\alpha}{\sqrt{v_t + \varepsilon}} \nabla_{param} E \tag{2.33}$$

$$param_{new} = param_{old} - \Delta x_t \tag{2.34}$$

Замість знаходження суми квадратів можна оновлювати експоненційне ковзне середнє (як для градієнту у momentum). Таким чином, дана величина буде враховувати зміну градієнту.

Існує також оптимізатор RMSProp, який майже ідентичний до AdaDelta за тим виключенням, що він рахує ковзне експоненційне середенє квадратів градієнтів не через суму, а через значення на кожному кроці (формула 2.35):

$$v_t = v_{t-1}\,\rho + (1 - \rho) \cdot \nabla_{param} E^2 \tag{2.35}$$

Одним із самих розповсюджених оптимізаторів є Adam. Він має гарні результати зведення помилки та стабільності. Він поєднує у собі ідеї накопичення імпульсу Momentum та врахування квадратів градієнтів у розрахунку RMSProp. Крім того додає адаптивну швидкість навчання, яка не викликає проблеми зникнення градієнту безпосередньо.

M(t) та V(t) – це величини значення першого та другого моменту (по суті градієнт та його квадрат в момент часу t). Вони розраховуються через експоненційне середнє ковзне (формули 2.36, 2.37):

$$m_t = m_{t-1}\beta_1 + (1 - \beta_1) \cdot \nabla_{param} E \quad (2.36)$$

$$v_t = v_{t-1}\beta_2 + (1 - \beta_2) \cdot \nabla_{param} E^{\ 2} \quad (2.37)$$

, де β_1, β_2 – коефіцієнти ковного експоненційного середнього які як правило беруться рівними 0.9 та 0,999 відповідно. Адаптивність швидкості навчання вводиться за допомогою наступних співвідношень (формули 2.38,2.39):

$$\hat{\mathrm{m}}_\mathrm{t} = \frac{\mathrm{m}_\mathrm{t}}{1 - \beta_1^\mathrm{t}} \quad (2.38)$$

$$\hat{\mathrm{v}}_\mathrm{t} = \frac{\mathrm{v}_\mathrm{t}}{1 - \beta_2^\mathrm{t}} \quad (2.39)$$

Параметр при цьому оновлюється за наступним законом (формула 2.40):

$$param_{new} = param_{old} - \alpha \frac{\hat{\mathrm{m}}_\mathrm{t}}{\sqrt{\hat{\mathrm{v}}_\mathrm{t}} + \varepsilon} \quad (2.40)$$

, де ε – невелике значення, що запобігає діленню на 0 (за замовченням $\frac{1}{10^{-8}}$).

Взагалом, існує багато різновидів оптимізаторів, які можуть давать деякий приріст у конкретних задачах. Проте, як правило навчання починають із перевірених та стабільних оптимізаторів, наприклад Adam, SGD, або SGD із Momentum [40, 41].

Висновки до розділу 2

Метою даного розділу було спрямувати напрямок подальших досліджень адже задач які вирішує машинний інтелект дуже багато. Таким чином, напрямком досліджень було обрано нейронні мережі прямого поширення, а саме згорткові нейронні мережі для вирішення задач класифікації та стискання графічних образів.

Зокрема, були розглянуті наступні питання:

- Принципи функціонування шарів, що задіяні при побудові згорткових нейронних мереж;
- Алгоритм навчання нейронних мереж Back Propagation на основі градієнтного спуску та особливості розповсюдження градієнту через шари згорткової мережі;
- Методи оптимізації функції втрат на основі градієнтного спуску.

3 РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір методів та технологій розробки

Існує багато готових рішень для машинного навчання, але здебільшого це бібліотеки (фреймворки, наприклад TensorFlow + Keras), а не програми з графічним інтерфейсом. В іншому випадку, якщо зустрічаються, власне програми, то їх функціонал суттєво програє зазначеним фреймворкам. Проте, програмне забезпечення з достатнім функціонал для вирішення задачі, може значно спросити роботу по навчанню мережі (робота із датасетами, будування топології, вибір параметрів ініціалізації, будування графіків навчання тощо). Прикладом може слугувати matlab із його окремим плагіном, що дозволяє реалізовувати нейромережеві архітектури (проте це не є його основним призначенням і тому встановлювати даний пакет заради лише одного розширення може виявитися недоцільним через великий обсяг, зайвого для вирішення конкретної задачі, функціоналу).

Таким чином, з огляду на існуючи рішення з яких можна почати розробку зазначеного програмного забезпечення було обрано мультимедійну платформу Unity за використанням мови програмування C#. Даний вибір був зроблений враховуючи наступні переваги:

1 Швидкість розробки (Мова програмування C# є досить простою, особливо в рамках платформи Unity, яка містить під собою потужний фрейморк який вже включає функціонал, що може буде використаний в рамках даної задачі. Докладніше нижчею);

2. Різні методи для роботи із зображеннями, текстурами, кольорами та матрицями. Можливість достатньо просто реалізовувати асинхронний та паралельний код. Математичні та інші бібліотеки, що є частиною фрейморку;

3. У повному обсязі реалізований компонетно-орієнтований підхід побудови проєкту, що є зручним для розробки інтерфейсу та інших частин програми;

5. Набір інструментів для створення та програмування інтерфейсу. Сюди

входять елементи інтерфейсу, та компоненти, які дозволяють будувати логіку взаємодії окремих елементів.

5. Compute Shaders. Це програми, що запускаються і виконуються на графічному процесорі. З огляду на те, що обчислення нейронних мереж – це набір тензорних операцій (які добре розпаралелюються), то такий спосіб розрахунків є значно ефективнішим у порівняні із звичайним процесором, адже на відміну від центрального процесора, де кількість потоків вираховується у десятках, графічний процесор має тисячі процесорів, хоч і з меншою частотою (але це компенсується їхньою кількістю) Compute Shaders використовують HLSL подібну мову програмування і дозволяють відносно просто проводити розрахунки на графічному процесорі. Даний інструмент доступний лише у рамках Unity. Основною його перевагою є універсальність, адже дані програми будуть запускатися на переважній більшості графічних процесорів, на відміну від специфічних технологій (наприклад Cuda, які працюватимуть лише на графічних процесорах Nvidia) [42];

Розробка програми для навчання нейронних мереж передбачає наявність наступних можливостей та компонентів:

- Система вводу/виводу. Сюди входить можливість завантажувати зображення датасету та зберігати/завантажувати файл мережі (налаштування мережі: топологія, параметри окремих шарів мережі, таких як ваги та зсуви, статистика і т.і), а також безпосередньо графічний інтерфейс який дозволяє викликати вищеперерахований функціонал;
- Програмна реалізація алгоритмів, необхідних для роботи та навчання нейронної мережі. Сюди входять алгоритми прямого та зворотного розповсюдження згорткових, повнозв’язних шарів, шарів субдискретизації, нормалізації тощо, а також функції втрат і алгоритми її оптимізації.
- Можливість зміни гіперпараметрів мережі за допомогою зручного інтерфейсу (топологія мережі, параметри ініціалізації, розмір батча,

кількість епох, вибір та налаштування оптимізатора тощо).

- Візуалізація даних про навчання та тестування моделі. Сюди входить: статистика (графік функції втрат, точність, час для кожної епохи), візуалізація активацій згорткових шарів під час навчання, можливість завантажувати окремі зображення для тесту та виводити відповідь мережі (список ймовірностей для кожного класу або вихідне зображення).

Враховуючи вищеперераховані вимоги до можливостей програмного забезпечення було створено проєктне рішення графічного інтерфесу, що зображений на рисунку 3.1.

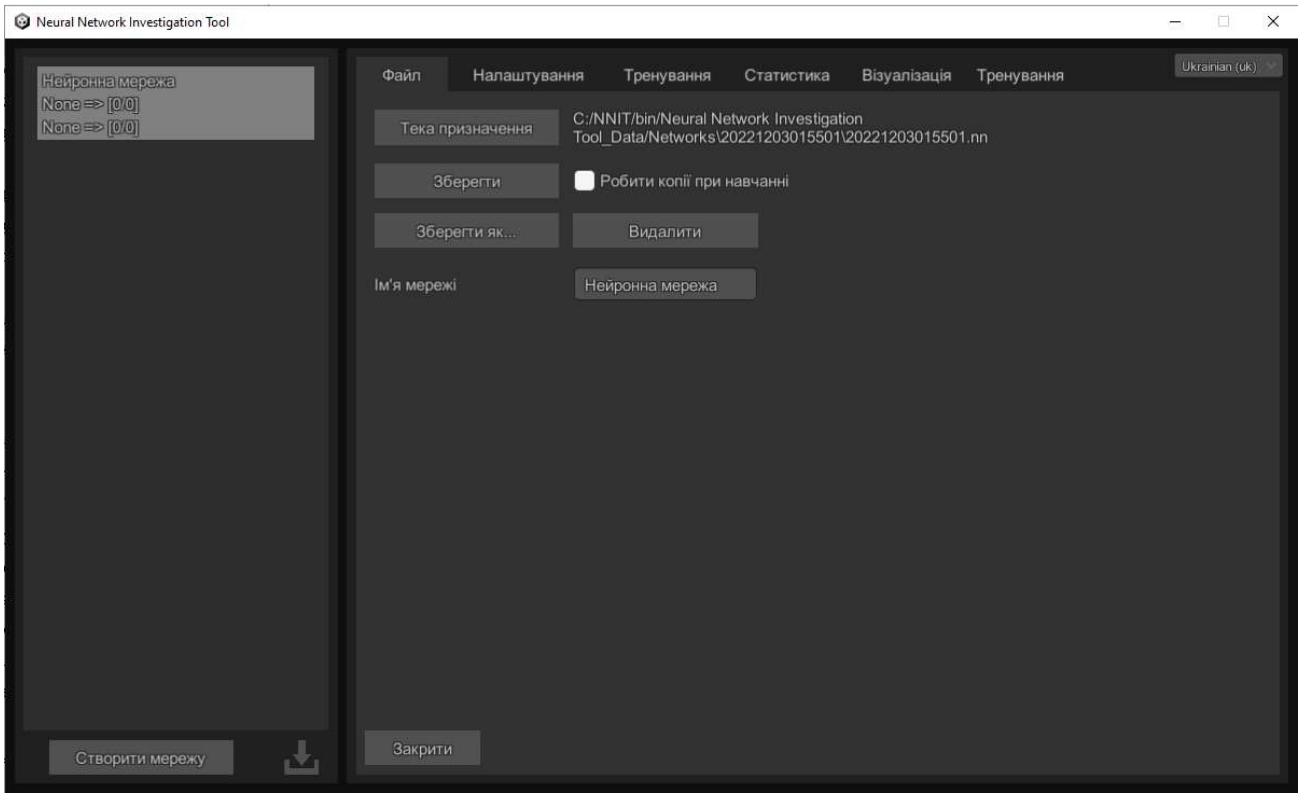


Рисунок 3.1 – Зовнішний вигляд графічного інтерфейсу програми.

Для зручності увесь функціонал розбитий на окремі вкладки навігація якими відбувається за допомогою альт-панелі зверху (також на ній присутня можливість змінити мову застосунку). Сліва знаходиться інтерфейс для завантаження створення та навігації між файлами. Елемент для вибору поточної

мережі (файлу мережі) також включає її назву та прогрес виконання поточних операцій (ініціалізації датасету, поточна епоха, поточний мінібатч, що виконується тощо).

Розробка програмного забезпечення відбувалася за допомогою Unity (рисунок 3.2),Visual Studio 2019 та мови програмування C#.

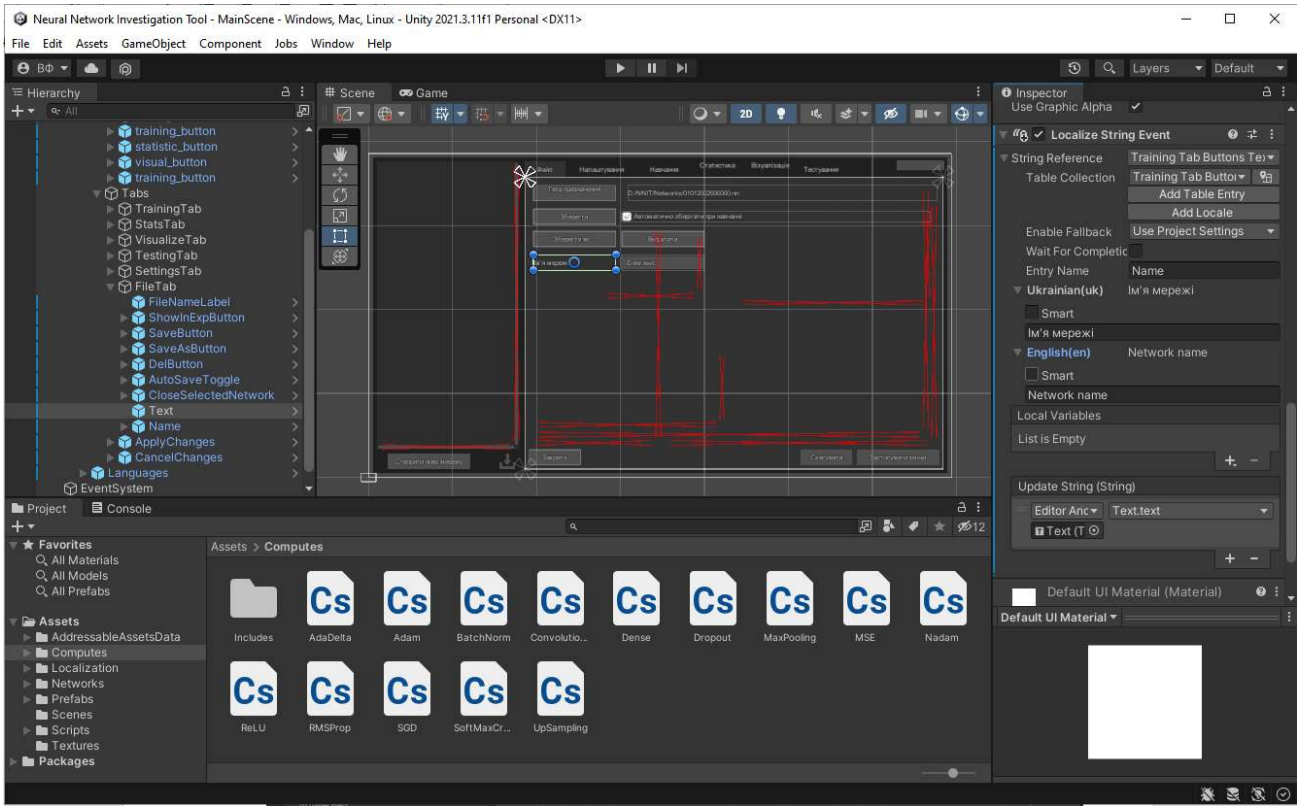


Рисунок 3.2 – Середовище розробки програмного забезпечення

Компонентно-орієнтований підхід до розробки дозволяє швидко нарощувати функціонал. Інтерфейс побудований ієрархічно. Кожен елемент має компоненти через які він взаємодіє з іншими (локалізація, логіка роботи елементу інтерфейсу, події і т.і).

Керуваннями даними компонентами на рівні програми та створення нових відбувається за допомогою C# скриптів. Для їх редагування використовувалась IDE Visual Studio 2019. На рисунку 3.3 зображена папка Assets завантажена для роботи у VS.

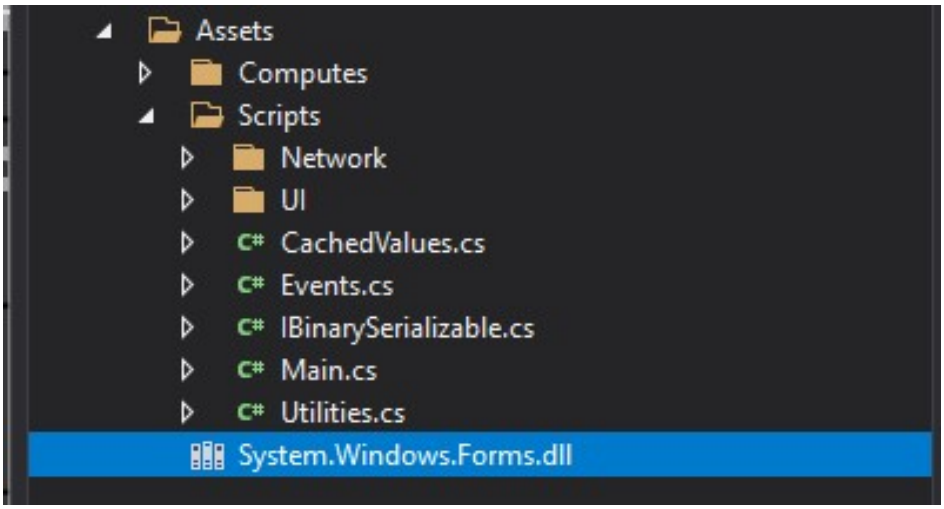


Рисунок 3.3 – Папка з ассетами проєкту

Усі скрипти розбиті на декілька глобальних категорій:

- Computes. Це Compute Shaders необхідні для роботи алгоритмів машинного навчання написані на HLSL - подібній мові програмування, яку використовує Unity;
- Scripts. Загальна директорія для усіх C# скриптів (компоненти, логіка їх взаємодії, алгоритми, інтерфейс і т.д);
- Network. В даній директорії знаходяться усі C# скрипти необхідні для функціонування нейронної мережі. Сюди входять: алгоритми машинного навчання (у двох екземплярах. Один для звичайного процесора, інший для графічного, що запускає Compute Shaders і завантажує в них необхідні дані), а саме логіка роботи окремих шарів нейронної мережі, оптимізатори на основі mini-batch градієнтного спуску та функції втрат для задач класифікації та регресії (в даному випадку -це стискання графічних образів);
- UI. Користувацький інтерфейс. В даній директорії зберігаються усі скрипти, які описують логіку взаємодії інтерфейсу та інших компонентів мережі (Вкладки, списки, графіки, таблиці і т.і). По суті це є системою вводу/виводу додатку. Деяка логіка поєднана із інтерфейсом з метою зменшити обсяг програми (програмного коду). MCV та йому подібні патерни проєктування використовувати для опису

не складного за структурою інтерфейсу або систему вводу/виводу може виявитися зайвим ускладненням процесу розробки.

- Cached Values. Скрипт який зберігає в собі глобальні змінні, до яких звертаються інші компоненти, а також змінні які задається із Inspector (Це інтерфейс середовища Unity, що дозволяє задавати значення полям класу зовні. Налаштовується для будь-яких типів даних).
- Events. Скрипт в якому зберігаються усі глобальні події. Сюди входять вибір/завантаження файлу, зміна вкладки, оновлення статистики і т.і.
- IBinarySerializible. Інтерфейс який дозволяє здійснювати сериалізацію/десиріалізацію бінарних даних. Іншими словами – це логіка збереження параметрів мережі у файл. Даний інтерфейс реалізують усі компоненти, що є частиною мережі і повинні зберігатися у файл для можливості повторного використання.
- Main. Не те саме, що main у класичному C#. Ініціалізує усі інші компоненти мережі при запуску застосунку.
- System.Windows.Forms. Стандартна бібліотека Windows. В даному випадку її використання зумовлено відсутністю у Unity стандартних рішень для створення діалогових вікон для завантаження та збереження файлів/папок.Для простоти уявлення того як повинні взаємодіяти компоненти програми варто побудувати блок-схему. На рисунку 3.4 зображені основні компоненти програми та їх взаємодія. В даному випадку блок-схема зосереджується навколо функціоналу необхідного для навчання нейронної мережі. Увесь інший функціонал так чи інакше пов’язаний із базовими компонентами та нарощувався з метою поліпшити зручність користування програмою

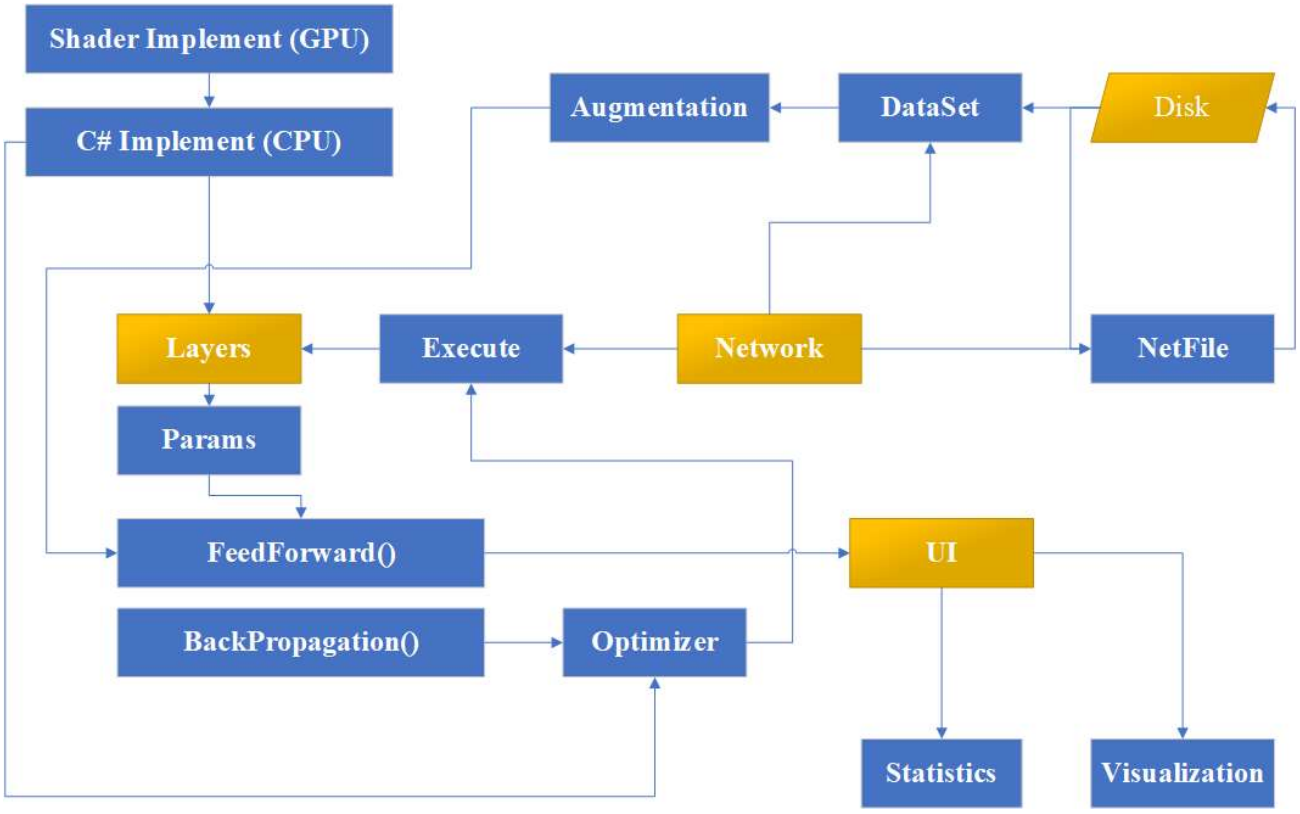


Рисунок 3.4 – Основні компоненти програмного забезпечення

3.2 Впровадження обраних проєктних рішень

3.2.1 Інтерфейс налаштування моделі нейронної мережі

Нейронна мережа має багато параметрів для налаштування, тому вони були розбиті на декілька окремих вкладок. На рисунку 3.5 зображений зовнішний вигляд інтерфейсу, що дозволяє здійснювати налаштування основних параметрів мережі, такий як топологія та спосіб у який виконуються розрахунки (за допомогою звичайного процесора або графічного). Також, зліва представлений список усіх доступних шарів мережі, що генерується автоматично із тих шарів, що мають програмний опис та інтерфейс вводу/виводу параметрів. Кожен шар, окрім вхідного та вихідного можна видалити або пермістити за допомогою графічного інтерфейсу. В свою чергу кожен окремий шар мережі має свої гіперпараметри для налаштування враховуючи специфіку їх функціонування.

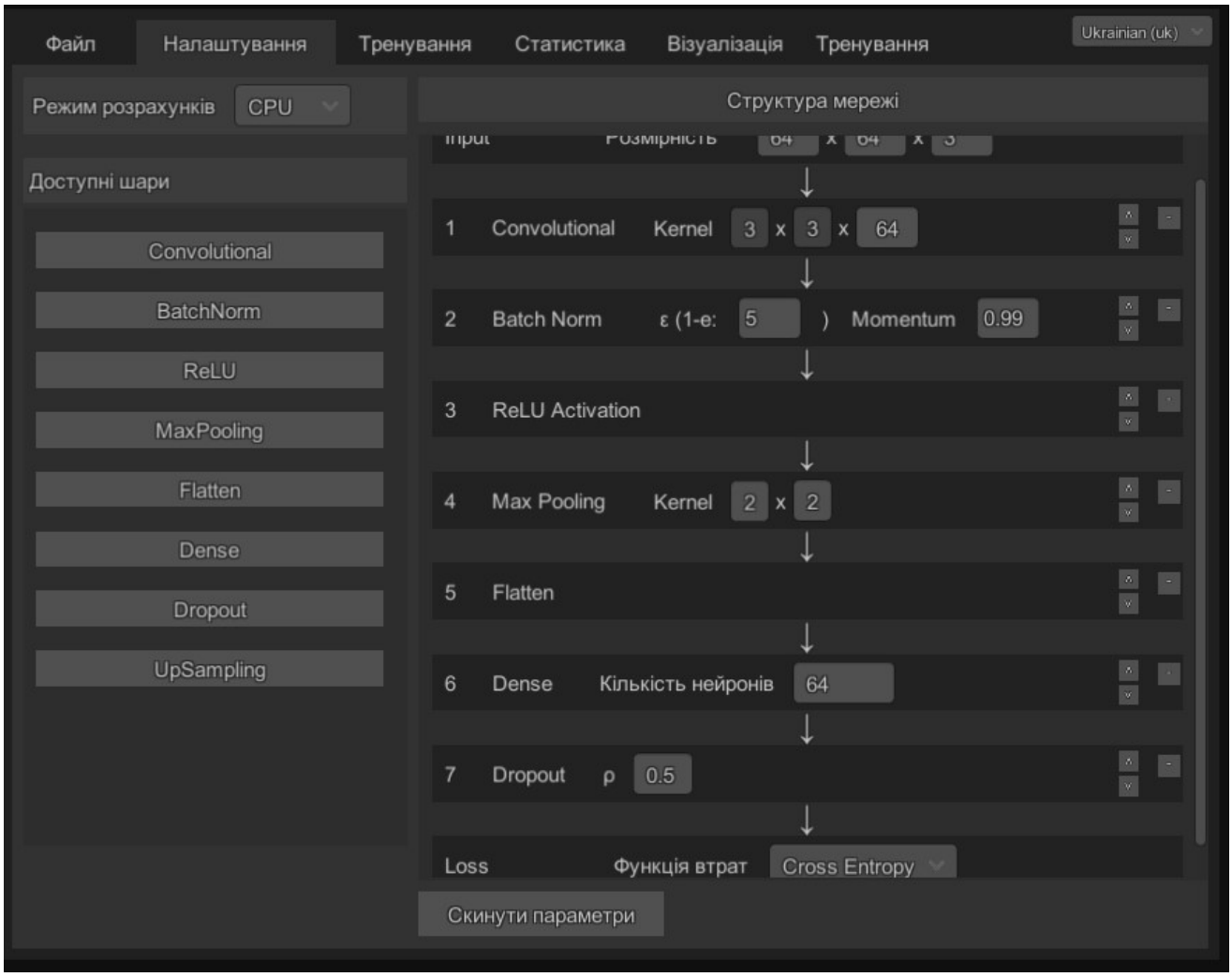


Рисунок 3.5 – Вкладка налаштувань моделі мережі

Параметри шарів що налаштовуються:

- Input. Розмір вхідного тензору та його глибина. Якщо це RGB зображення то глибина буде три (на кожну кольорову складову, прозорість не використовується), якщо це чорно-біле зображення (наприклад рукописні цифри із MNIST) то канал буде один.
- Convolutional. Кількість згорткових фільтрів. Розмір фільтру завжди дорівнює 3x3. Наразі інші розмірі фультру як правило не використовується);
- Batch Norm. Константа для запобігання діленню на нуль та коефіцієнт для ковзного екпоненційного середнього, що використовується для розрахунку параметрів необхідний під час тестування мережі;
- Max Pooling. Розмір та крок ковзного вікна. Крок дорівнює розміру;

- Dense. Кількість нейронів у повнозв’язному шарі;
- Dropout. Ймовірність того, що нейрон залишиться активованим на кожному кроці;
- UpSampling. Множник масштабу. Визначає у скільки раз вхідний тензор буде збільшений по ширині та по висоті;
- Loss. Фукнція втрат. Soft Max + Cross Entropy для задач класифікації (останнім повинен йти повнозв’язний шар із кількістю нейронів, що відповідає кількості класів) та середньоквадратичне відхилення, MSE для задач регресії (останнім повинен йти шар виходом якого є тензор, що представляє із себе кольорове зображення, тобто має глибину 3).

Кожен параметр має оптимальне значення за замовчуванням, що в більшості випадків залишається незмінним до моменту валідації гіперпараметрів. Це дозволяє побудувати топологію досить швидко (швидше ніж за допомогою програмного коду).

Усі алгоритми машинного навчання, що використовуються при роботі шарів нейронної мережі використовують формули наведені у попередньому розділі (формули 2.1 - 2.40). Лістинг програмної реалізації на основі Compute Shaders для кожного із зазначених шарів мережі наведений у Додатку Б.

3.2.2 Керування навчанням моделі нейронної мережі

На рисунку 3.6 зображений зовнішній вигляд інтерфейсу для керування навчанням мережі. Дана вкладка налічує найбільшу кількість функціоналу, і поділяється на три основні блоки: налаштування датасету (блок №1), налаштування оптимізатора (блок №2), аугментація навчальних даних (блок №3), та панель запуску навчання/ініціалізації (блок №4).

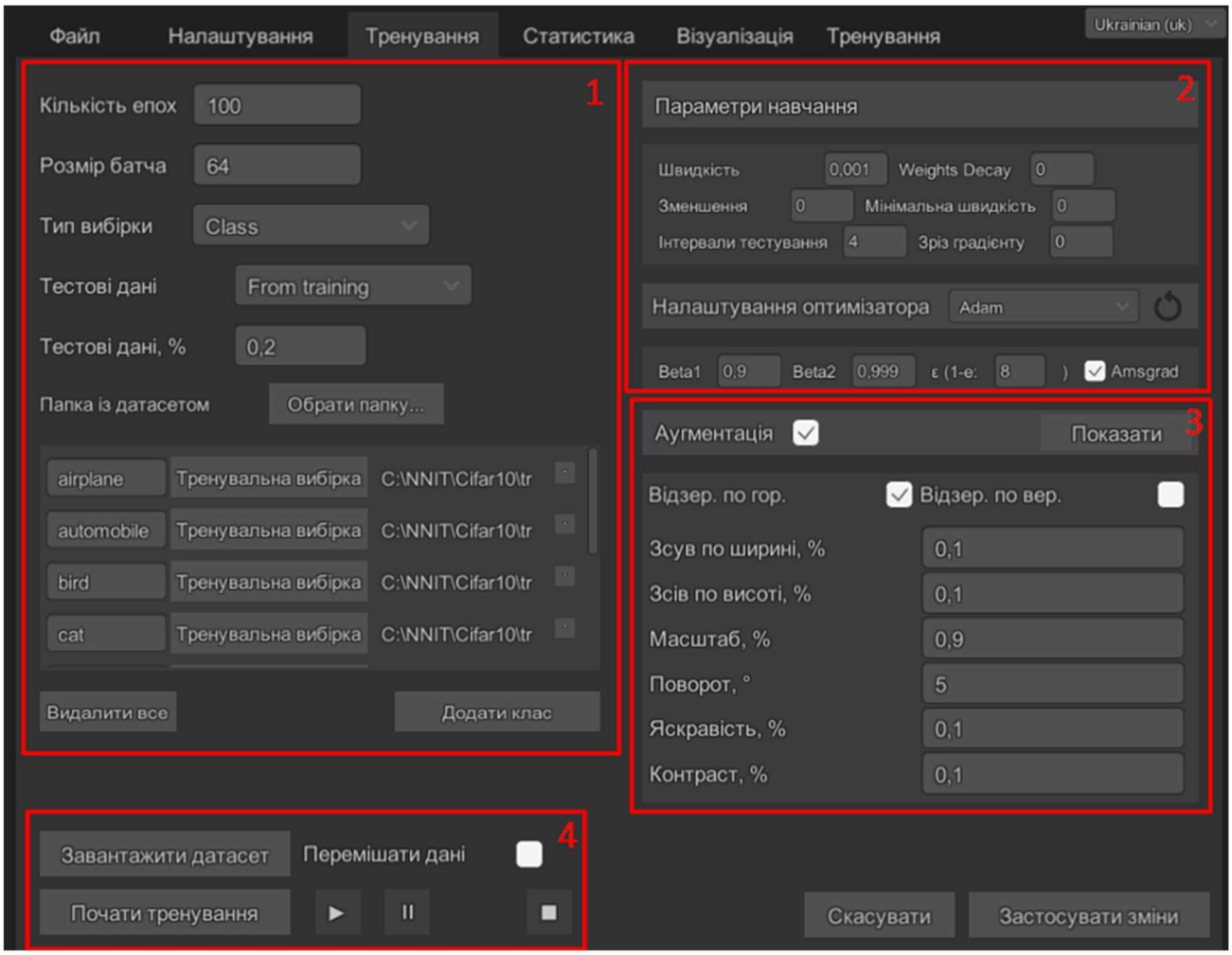


Рисунок 3.6 – Інтерфейс керування навчанням моделі нейронної мережі

Налаштування датасету включає наступні параметри:

- розмір міні-батчу, який впливає на кількість спостережень, які приймають участь в одній ітерації циклу навчання;
- кількість начальних епох, тобто проходів по всьому датасету;
- тип датасету («класифікація», тобто промарковані зображення, або «Зображення», тобто на виході мережі буде те саме зображення, що і на вході;
- режим в якому обираються тестові дані. Передбачає два різних налаштування: «Separate», при якому тестові дані мають окрему директорію (рисунок 3.7, а) та, «From training», при якому тестові дані генеруються із навчальних (при цьому з’являється додаткове поле вводу у якому можна обрати % навчальних даних який буде використаний,

рисунок 3.7, б);

- вибір директорії для навчальних та тестових даних. Шлях до датасету можна згенерувати автоматично (рисунок 3.7, в), якщо дані структуровані, або для кожного класу обрати директорію в якій зберігаються навчальні/тестові зображення.

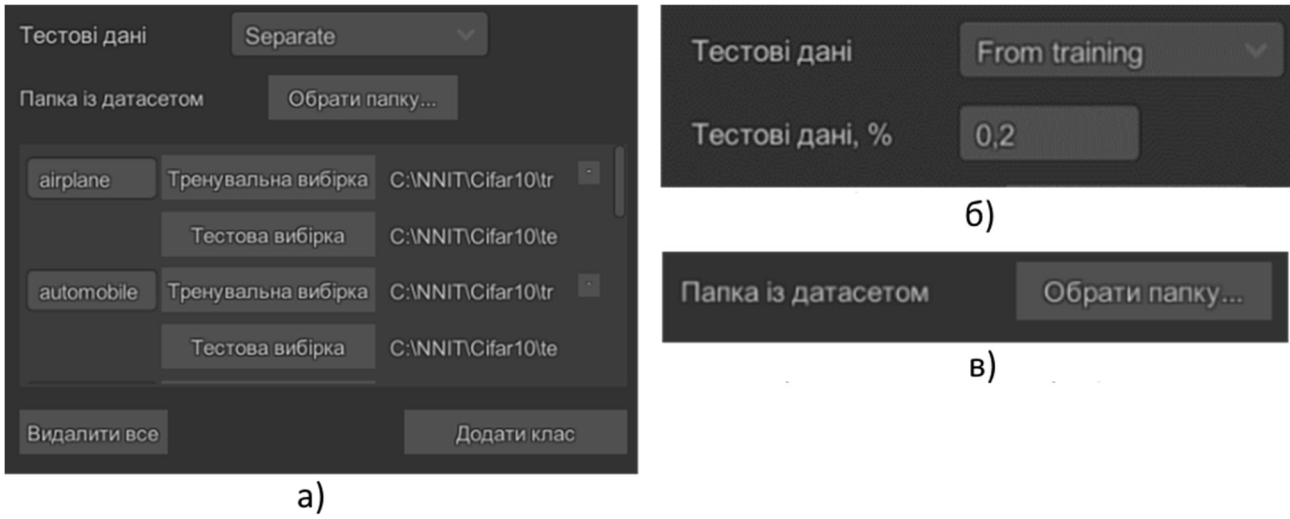


Рисунок 3.7 – Завантаження датасету: тестові дані окремо (а), згенеровані з навчальних (б), автоматична генерація шлахів (в)

Одним із важливих елементів навчання є правильний вибір алгоритму оптимізації функції втрат. Існує багато різних оптимізаторів на основі градієнтного спуску (деякі з них наведені у розділі 2). Як правило для перевірки сходимості помилки в першу чергу використовують SGD (саме він виставлений за замовчуванням) або Adam. Інші оптимізатори випробовують від час валідації гіперпараметрів.

Загальними, для всіх оптимізаторів, параметрами навчання моделі мережі, що регулюються є:

- швидкість навчання;
- weight decay (коефіцієнт пропорційності для зменшення числових значень параметрів мережі);
- зменшення швидкості навчання за експонентою (адаптивна швидкість навчання іноді допомогає досягти кращих результатів сходимості

помилки);

- мінімальний порог швидкості навчання (до якого значення швидкість навчання може зменшуватися);
- інтервали тестування (валідації). Перевірка точності моделі на тестових даних через певний інтервал (кількість епох) навчання;
- зріз градієнту. Для обмеження амплітуди зміни параметрів мережі. Усуває ефект «вибуху градієнту», якщо інші засоби не допомогли.

Крім того, при виборі оптимізатора з’являється налаштування параметрів кожного з них.

Ефективним способом збільшити властивість моделі мережі узагальнювати вхідні дані є аугментація спостережень датасету. Даний підхід дозволяє штучно збільшити розмір навчальної вибірки за рахунок генерації нових зображень на основі оригінального.

Аугментація даних віключає наступні параметри, що налаштовуються:

- Віддзеркалення зображення по ширині та/або по висоті;
- Зсув по ширині та/або по висоту. При цьому зберігається масштаб оригінального зображення;
- масштабування зображення відносно центру (кроп);
- поворот зображення на довільний кут (генерується два зображення для додатного та від’ємного значення кутів). Зображення, при цьому зменшується у масштабі в залежності від кута, для уникнення пустих областей, що виникають при його повороті;
- яскравіть. Зміна яскравості зображення на певний відсоток (генерується два зображення, із збільшеною яскравістю та із зменшеною);
- контраст. Зміна контрасту зображення на певний відсоток (генерується два зображення, із збільшеним контрастом та із зменшеним).

Приклад застосування аугментації (параметри зображені на рисунку 3.8, а) до зображення наведений на рисунку 3.8, б.

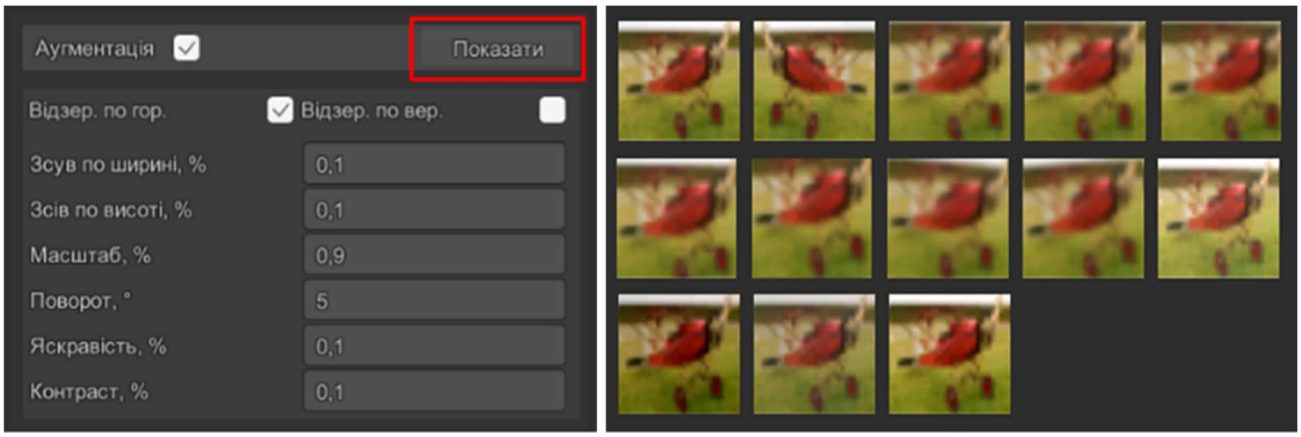


Рисунок 3.8 – Аугментація зображення 32x32 із параметрами (а) та візуалізацією результату (б)

Візуально знайти відмінності між окремими зображення на рисунку досить складно, просто якщо аналізувати їх попіксельно вони матимуть відмінний розподіл. Саме даний ефект дозволяє моделі мережі отримувати при навчанні кращі можливості із узагальнення, адже вона має змогу аналізувати одне й те саме зображення під різними «кутами» знаходячі більш складні патерни для відтворення.

Панель запуску навчання надає можливість почати навчання, призупинити, відновити та перервати. Ініціалізація датасеті відбувається у відповідності до обраних параметрів у блоці датасету (розмір батча, директорії де зберігаються зображення і т.і). Дані для навчання можна перемішати (в тому числі аугментовані дані).

3.2.3 Статистика навчання. Візуалізація

Важливим елементом навчання моделі нейронної мережі є вивід статистики. Сюди в першу чергу входить графік функції втрат. Також це може бути точність, час витрачений на одну епоху тощо. На рисунку 3.9 зображена вкладка виводу статистики навчання. Зліва знаходиться графік функції помилки для обраної ехопи.

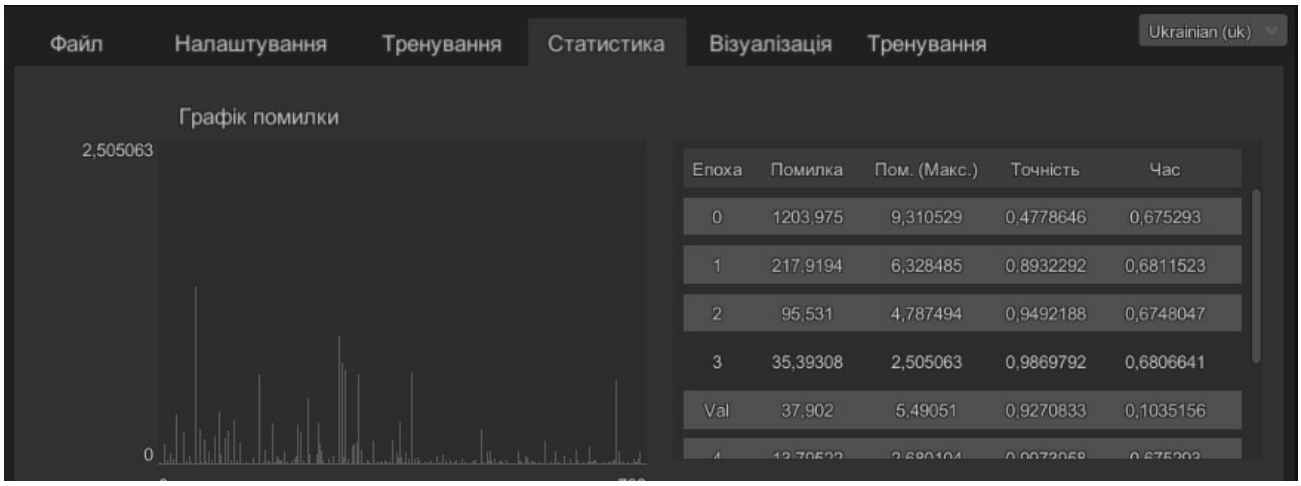


Рисунок 3.9 – Форма виведення статистики навчання моделі мережі

Справа перелік епох навчання у вигляді списку-таблиці, де можна спостерігати сумарну помилки для конкретної епохи, максимальну помилку, тончість (у задачах класифікації, визначається як відношення правильних відповідей до загальної кількості спостережень. Правильна відповідь – це те і якої елемент векторі ймовірності для правильного класу набуває максимального значення). Також, можна обрати будь які епоху із списку, а також завантажити копію мережі для даної епохи якщо вона наявна.

Існують також інші способи візуалізувати дані, при навчанні нейронної мережі. На відміну від повнозв’язних шарів, де приховані шари не несуть у собі корисної для людини інформації (працюють за принципом чорного ящику) згорткові шари, а саме принцип їх роботи, є інтуітивно зрозумілим, тому візуалізація активацій згорткових шарів може бути корисною для розуміння того, яким чином модель навчається узагальнювати дані.

На рисунку 3.10 зображена візуалізація активацій згорткових шарів мережі класифікації графічних образів.

Рисунок 3.10 – Візуалізація активацій згорткових шарів (Feature Maps)

Візуалізація починається з вхідного шару мережі. Як видно з рисунку, перші шари мережі реагують на прості патерни, такі як перепади яскравості або лінії і добре окреслюють контур об’єкта якщо дані патерни яскраво виражені.

3.2.4 Загальні налаштування моделі

Окрім вищеперерахованого функціоналу наявна вкладка для роботи із файлом мережі (рисунок 3.11).

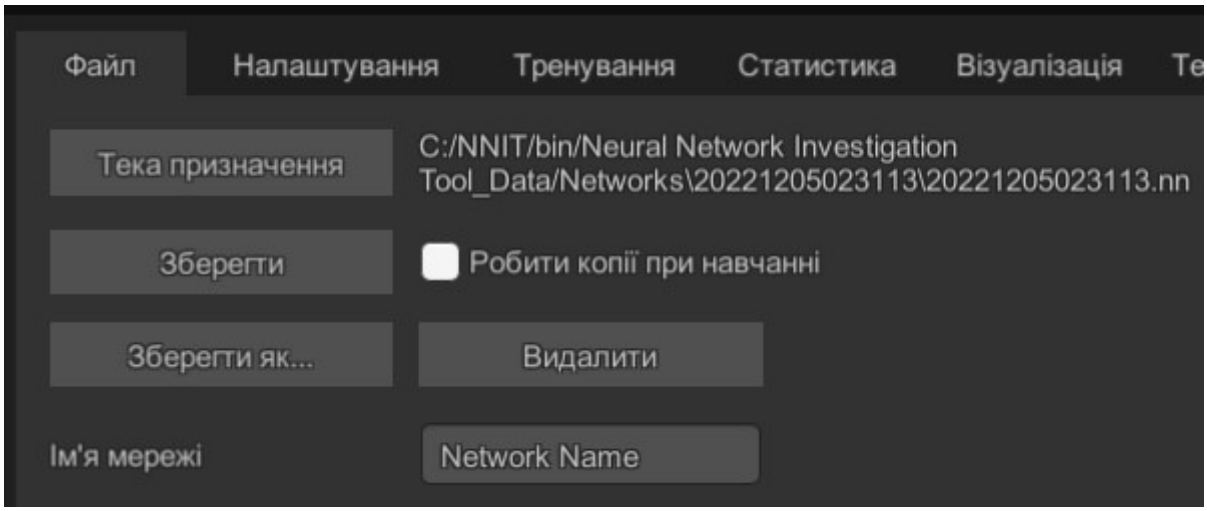


Рисунок 3.11 – Вкладка налаштувань файлу мережі

За допомогою даної вкладки можна:

- задати ім’я мережі (буде відображатися у списку мереж);
- відкрити теку із розташуванням файлу мережі;
- зберегти параметри мережі;
- зберегти модель у новий файл;
- видалити файл моделі;
- робити копії параметрів моделі під час навчання (опціонально, необхідно-для можливості повернутися до більш вдалої конфігурації).

Останньою є вкладка, що дозволяє проводити тестування моделі (за допомогою тестових даних, що визначаються на вкладці навчання або на інших даних які можна завантажити окремо).

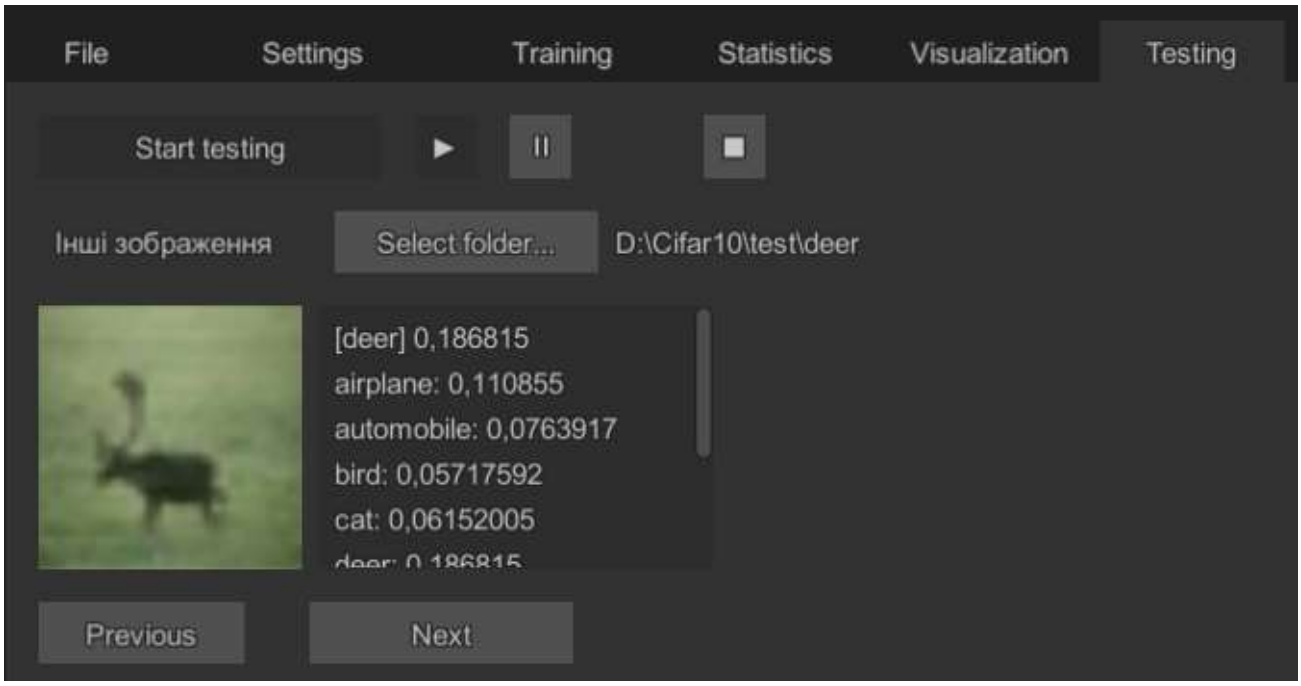


Рисунок 3.12 – Параметри тестування моделі мережі

Для тестування сторонніх даних необхідно обрати папку із зображеннями. Навігація по зображенням відбувається за допомогою відповідних кнопок. Усі зображення перетворюються у вигляд придатний для розповсюдження мережею. Відповідь формується в залежності від типу мережі (список класів та імовірностей для кожного із них, якщо це мережа класифікації та зображення якщо виходом мережі є зображення) [43, 44].

3.3 Тестування програмного забезчення

Так як усі алгоритми машинного навчання були реалізовані з нуля необхідно переконатися у правильності їх роботи. Для цього доцільно навчити модель вирішувати відносно складну задачу. В якості такої задачі було обрано класифікацію зображень із датасету Cifar 10. Даний датасет налічує 60000 кольорових зображень 32x32 пікселі для кожного із 10 представлених у ньому класів.

Для розв’язання даної задачі була обрана топологія мережі, яка включає у собі всі наявні структурні блоки. Згорткова частина моделі складається із блоків представлених на рисунку 3.13.

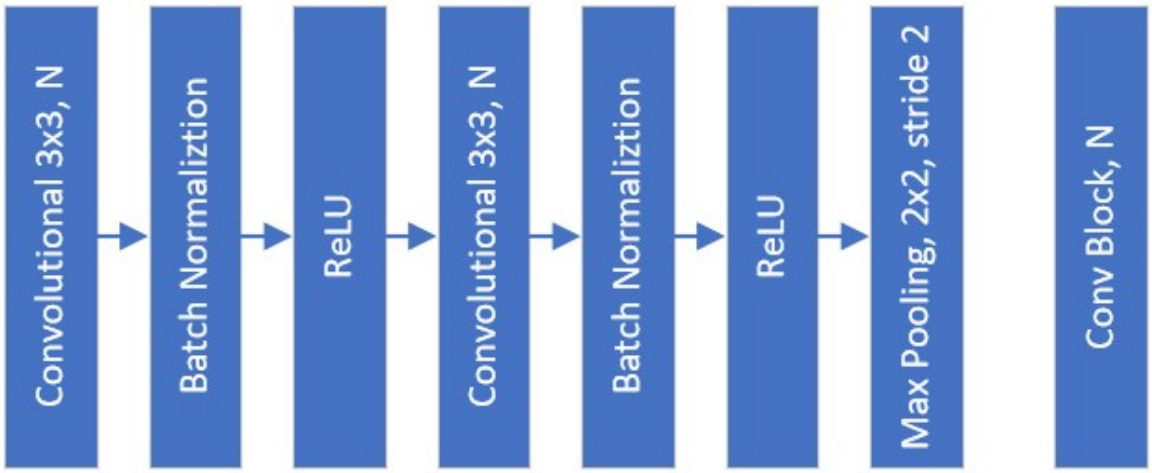


Рисунок 3.13 – Структурний блок згорткової частини мережі

, де N – кількість згорткових фільтрів для згорток даного блоку, що позначається через Conv Block, N. Загальна структура мережі при цьому зображена на рисунку 3.14.

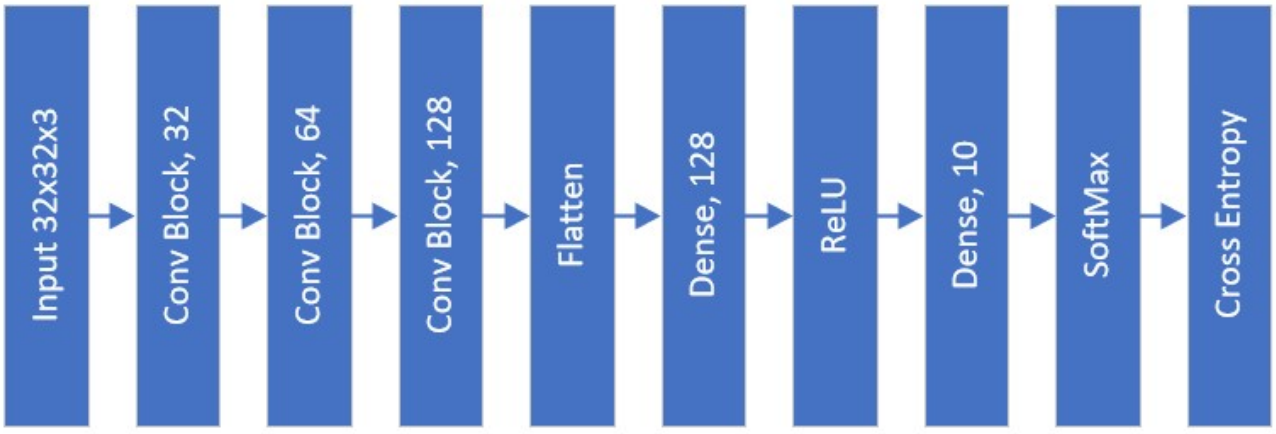


Рисунок 3.13 – Топологія мережі класифікації

Мережа досягла максимальної точності на восьмій епосі у 77% на тестових даних (рисунок 3.14). Для даної топології мережі це є нормальним показником. Для збільшення точності моделі можна застосувати Dropout для першого повнозв’язного шару але для тесту цього достатньо.

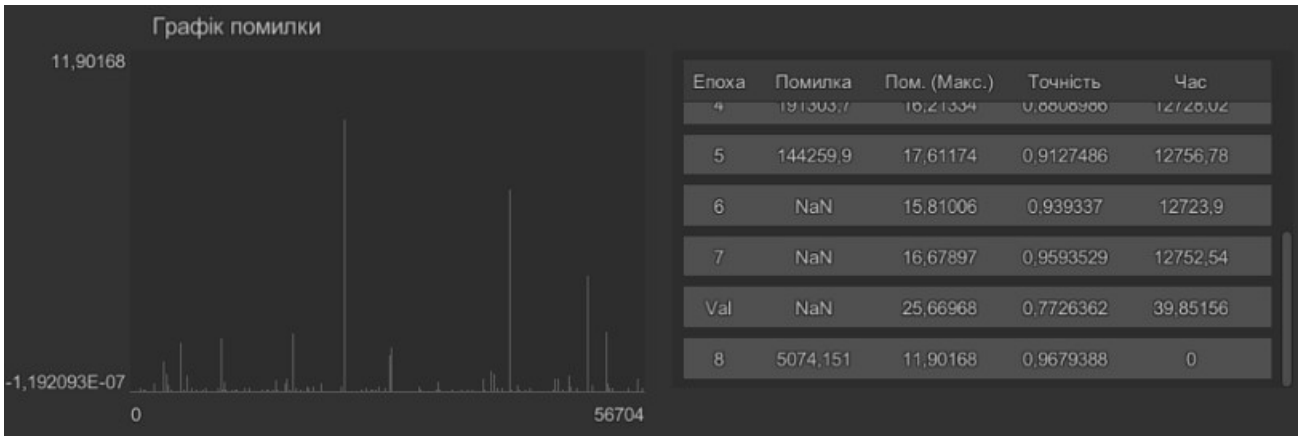


Рисунок 3.14 – Тестування моделі мережі на датасеті Cifar-10

Також, в даному випадку сходимість помилки так швидко (на восьмій епосі) зумовлена використанням Batch Normalization та аугментації вхідних даних (були застосовані усі доступні параметри окрім вертикального відзеркалення, що значно збільшило кількість навчальних даних) [45].

3.4 Розробка моделі стискання графічних образів

Раніше була згадана модель мережі Bottle Neck, основна ідея якої полягає у тому, що на виході мережі очікуються ті самі дані, що і на вході, проте деяка частина мережі повинна мати меншу розмірність у порівняні із вхідними даними. За рахунок цього мережу можна розбити на дві частини (кодер та декодер). Перша частина буде здійснювати репрезентацію зображення у стислій формі, а друга частина відновлювати його.

Є два способи зменшити розмірність тензору: зменшити його глибину за допомогою операції згортки із меншою кількістю фільтрів ніж є у ньому, та зменшити ширину та висоту за допомогою операцію Max Pooling. Крім того, потім необхідно відновити вихідну розмірність зображення, для цього знов можна використовувати згортки та операцію Up-Sampling. Таким чином, один із можливих варіантів побудови такої мережі запропонований на рисунку 3.15.

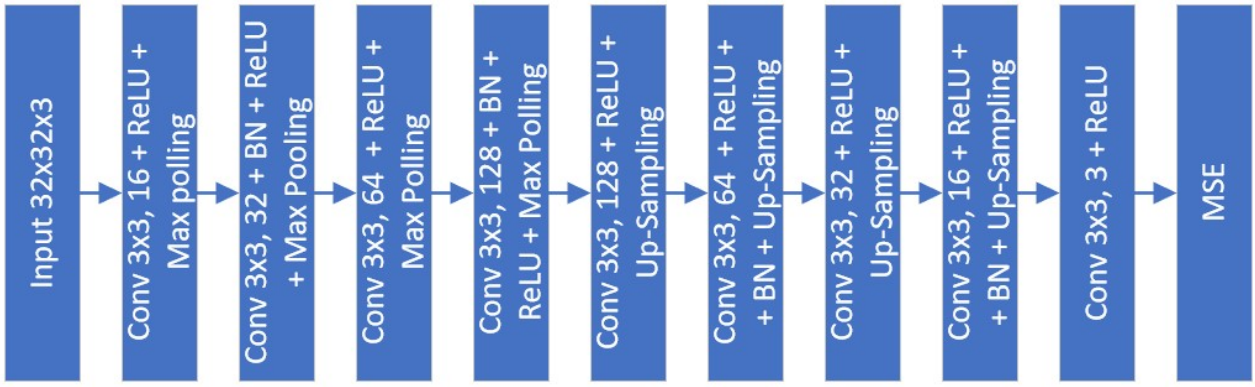


Рисунок 3.15 – Топологія мережі для стискання графічних образів

Для навчання даної моделі так само були використані зображення із датасету Cifar 10. На рисунку 3.16 зображений результат роботи такої мережі у вігляді декількох пар зображень. Зліва оригінальне зображення, справа реконструйоване.

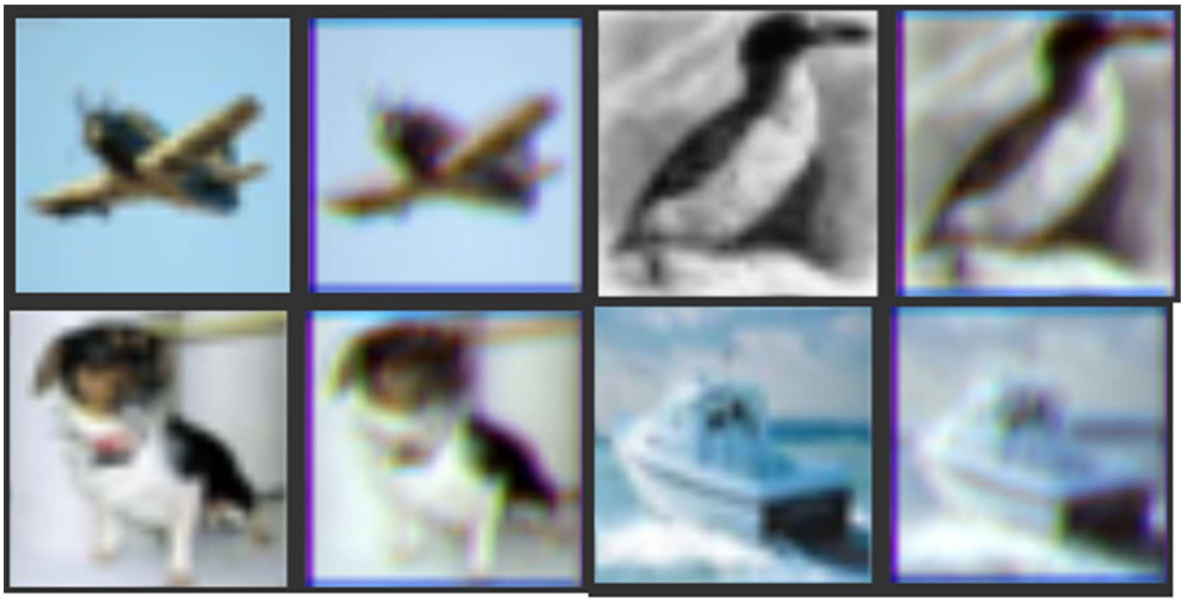


Рисунок 3.16 – Результат роботи мережі стискання зображень

Висновки до розділу 3

У даному розділі був описаний процес розробки програмного забезпечення для створення моделей нейронних мереж класифікації та стискання графічних образів. Зокрема було:

- Обрано технології та методи розробки;
- Розроблені проєктні рішення для обраної задачі;

- Описано процес впровадження технічних рішень;
- Перевірено правильність роботи описаних алгоритмів машинного навчання на задачі класифікації;
- Розроблено модель мережі для стискання графічних образів.

ВИСНОВКИ

В даній магістерській роботу було здійснено розроблення програмного забезпечення із графічним інтерфейсом, що дозволяє здійснювати навчання нейронних мереж прямого розповсюдження (згорткових мереж), а також дослідження навчання мережі стиснення зображень.

Робота включала наступні основні етапи:

1. Дослідження поняття нейронної мережі, аналіз стану розвитку даної галузі на сьогоднішній день, особливості функціонування нейронних мереж, її переваги та недоліки у порівнянні із звичайними алгоритмами.
2. Вибір напрямку подальших досліджень, а саме згорткових нейронних мереж класифікації та скискання графічних образів. Опис алгоритмів машинного навчання необхідних для функціонування даного типу мереж у різних задачах. Сюда входять алгоритми функціонування окремих шарів мережі, функцій втрат та методів оптимізації функції втрат на основі градієнтного спуску.
3. Розроблення програмного забезпечення, що дозволяє здійснювати навчання моделей нейронних мереж для класифікації та стискання графічних образів: вибір методів та технологів розробки, розробка та впровадження оптимальних проєктних рішень;
4. Перевірка роботи алгоритмів машинного навчання у задачі класифікації. Для обраної топології мережі за вхідних даних була отримана точність у 78% на тестових даних, що відповідає даній моделі.
5. Створення та навчання моделі мережі для стискання графічних образів. Була обрана топологія згорткової нейронної мережі на основі архітектури Bottle Neck (Кодер – Декодер/Реконструктор). Мережа успішно навчилася стискати зображення 32x32 (із датасету Cifar10).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Philip D. Wasserman. Neural Computing: Theory and Practice / Philip D. Wasserman. – Van Nostrand Reinhold, 1989. – 230 с.

2. Simon Haykin. Neural Networks: A Comprehensive Foundation / Simon Haykin. – Hamilton, Ontario , Canada: Prentice-Hall Of India Pvt. Limited, 1999. – 842 с. – (2).

3. Artificial Neural Network | SpringerLink [Електронний ресурс] // Springer, Boston, MA. – 2003. – Режим доступу до ресурсу: https://link.springer.com/chapter/10.1007/978-1-4615-0377-4_5.

4. Zhihua Zhang. Multivariate Time Series Analysis in Climate and Environmental Research / Zhihua Zhang. – College of Global Change and Earth System Science: Springer, 2017. – 287 с.

5. Yu-chen Wu. Development and Application of Artificial Neural Network [Електронний ресурс] / Yu-chen Wu, Jun-wen Feng // Springer. – 2017. – Режим доступу до ресурсу: https://link.springer.com/article/10.1007/s11277-017-5224-x.

6. Класифікація нейронних мереж та їх властивості. [Електронний ресурс] – Режим доступу до ресурсу: https://dl.nure.ua/pluginfile.php/634/mod_resource/content/2/content/content1.html.

7. Одношаровий перцептрон [Електронний ресурс] – Режим доступу до ресурсу: https://studfile.net/preview/5083084/page:2/.

8. Leonardo Noriega. Multilayer Perceptron Tutorial [Електронний ресурс] / Leonardo Noriega // Staffordshire University. – 2005. – Режим доступу до ресурсу: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.608.2530&rep=rep1&type=pdf.

9. What are Convolutional Neural Networks? | IBM [Електронний ресурс] // IBM Cloud Education. – 2020. – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/convolutional-neural-networks.

10. Teuvo Kohonen. Self-Organizing Maps / Teuvo Kohonen. – Heidelberg: Springer Science & Business Media, 2012. – 502 с.

11. Broomhead, D. S. RADIAL BASIS FU NCTIONS, MULTI-VARIABLE FUNCTIONAL INTERPOLATION AND ADAPTIVE NETWORKS / Broomhead, D. S, Lowe, David. – London, 1988. – 39 с.

12. Т.А. Самолюк. Нейромережі GAN у створенні нових моделей [Електронний ресурс] / Т.А. Самолюк. – 2019. – Режим доступу до ресурсу: http://dspace.nbuv.gov.ua/bitstream/handle/123456789/168482/14-Samolyuk.pdf?sequence=1.

13. Dr. John J. Hopfield. Hopfield network [Електронний ресурс] / Dr. John J. Hopfield // Scholarpedia. – 1977. – Режим доступу до ресурсу: http://scholarpedia.org/article/Hopfield_network.

14. Колесницький Олег. КОМПРЕСІЯ ЗОБРАЖЕНЬ ЗА ДОПОМОГОЮ НЕЙРОННОЇ МЕРЕЖІ ПРЯМОГО ПОШИРЕННЯ [Електронний ресурс] / Колесницький Олег, Міщанчук Андрій – Режим доступу до ресурсу: http://ir.lib.vntu.edu.ua/bitstream/handle/123456789/13332/38-39.pdf?sequence=1.

15. Visualization of Convolution layers in a CNN and its importance [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://medium.com/analytics-vidhya/visualization-of-convolution-layers-in-a-cnn-and-its-importance-44d00ad4eb70.

16. Згорткова нейронна мережа – просте пояснення CNN та її застосування [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://evergreens.com.ua/ua/articles/cnn.html.

17. JPEG [Електронний ресурс] – Режим доступу до ресурсу: https://hmn.wiki/uk/JPEG#Discrete_cosine_transform.

18. Image Compression using Backprop [Електронний ресурс] // 2021 – Режим доступу до ресурсу: https://web.archive.org/web/20070828112920/http://neuron.eng.wayne.edu/bpImageCompression9PLUS/bp9PLUS.html.

19. What Are “Bottlenecks” in Neural Networks? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.baeldung.com/cs/neural-network-bottleneck.

20. CNN Tensor Shape Explained [Електронний ресурс] – Режим доступу до ресурсу: https://deeplizard.com/learn/video/k6ZF1TSniYk.

21. Convolutional Neural Networks [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/convolutional-neural-networks.

22. What is the ReLU layer in CNN? [Електронний ресурс] – Режим доступу до ресурсу: https://www.quora.com/What-is-the-ReLU-layer-in-CNN.

23. Zero Padding in Convolutional Neural Networks [Електронний ресурс] – Режим доступу до ресурсу: https://deeplizard.com/learn/video/qSTv_m-KFk0.

24. Convolutional Neural Networks, Explained [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

25. Autoencoder: Downsampling and Upsampling [Електронний ресурс] – Режим доступу до ресурсу: https://kharshit.github.io/blog/2019/02/15/autoencoder-downsampling-and-upsampling.

26. Different Types of CNN Architectures [Електронний ресурс] – Режим доступу до ресурсу: https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/.

27. Introduction to Loss Functions [Електронний ресурс] // DataRobot. – 2018. – Режим доступу до ресурсу: https://www.datarobot.com/blog/introduction-to-loss-functions/.

28. Convolutional Neural Networks (CNN): Softmax & Cross-Entropy [Електронний ресурс] – Режим доступу до ресурсу: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-softmax-crossentropy.

29. Mean squared error [Електронний ресурс] – Режим доступу до ресурсу: https://peltarion.com/knowledge-center/modeling-view/build-an-ai-

model/loss-functions/mean-squared-error.

30. ReLU Derivative in Back Prop [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/questions/42042561/relu-derivative-in-backpropagation.

31. Understanding Backpropagation Algorithm [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd.

32. How do GD, Batch GD, SGD, and Mini-Batch SGD differ? [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://datascience.stackexchange.com/questions/53870/how-do-gd-batch-gd-sgd-and-mini-batch-sgd-differ.

33. What is the different between MSE error and Cross-entropy error in NN [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://susanqq.github.io/tmp_post/2017-09-05-crossentropyvsmes/.

34. Backpropagation through a fully-connected layer [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://eli.thegreenplace.net/2018/backpropagation-through-a-fully-connected-layer/..

35. Back Propagation in Fully Connected Layer [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.adityaagrawal.net/blog/deep_learning/bprop_fc.

36. Backpropagation In Convolutional Neural Networks [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/.

37. A gentle explanation of Backpropagation in Convolutional Neural Network (CNN) [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://medium.com/@ngocson2vn/a-gentle-explanation-of-backpropagation-in-convolutional-neural-network-cnn-1a70abff508b.

38. Pooling Layer [Електронний ресурс] – Режим доступу до ресурсу:

https://leonardoaraujosantos-gitbook-io.translate.goog/artificial-inteligence/machine_learning/deep_learning/pooling_layer?_x_tr_sl=auto&_x_tr_tl=uk&_x_tr_hl=uk.

39. Backpropagation Through Max-Pooling Layer [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://leimao.github.io/blog/Max-Pooling-Backpropagation/.

40. Various Optimization Algorithms For Training Neural Network [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6.

41. Optimization Algorithms in Neural Networks [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html.

42. Compute Shaders [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/class-ComputeShader.html.

43. Unity3D [Електронний ресурс] – Режим доступу до ресурсу: https://unity.com/.

44. Visual Studio IDE [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/.

45. How to Develop a CNN From Scratch for CIFAR-10 Photo Classification [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/.

ДОДАТОК А

1 PRINCIPLES OF FUNCTIONING, CONSTRUCTION AND FEATURES OF ARTIFICIAL NEURAL NETWORKS

1.1 The structure and working principles of neural networks

1.1.1 Reference model for building neural networks

Humanity, engineers in particular, quite often take an example from nature to create new technical solutions based on them. Artificial neural networks are not an exception - the study of the principles of functioning of biological neural networks (actually, the brains of people and animals) made it possible to describe a mathematical model that became the basis for the creation of new areas of computer intelligence.

The creation of artificial neural networks owes to research in the neurobiology branch of science, and in some cases, we can find behavior similar to the work of the human brain in the results of the work of artificial neural networks. However, the very frequent comparison of the processes occurring in the brain and the principle embedded in artificial networks is useless, because the real connection between biology and computer intelligence is very weak, and often causes a number of misunderstandings. In addition, our knowledge of the principles of the functioning of the biological brain is not fundamental, which indicates the impossibility of reproducing the processes embedded in the brain as an understandable mathematical model.

Even without forgetting the above, it is useful to know the basic principles involved in the work of the biological nervous system, because artificial neural networks try to approach biological networks in terms of efficiency when solving tasks that humans and animals solve without problems [1].

Research on artificial neural networks is mainly related to the fact that the way a human or animal nervous system processes information is very different from the way computers do it.

If you simplify, the brain can be imagined as a very complex, non-linear

structure, the main components of which are neurons, which have the property of self-organization, which in turn allows you to perform tasks that are difficult for ordinary computing equipment. Examples can be classification (recognition of images, sounds, smells, etc.), motor functions (a common action for us, for example, walking is the use of a huge number of muscles in the correct sequence, which is what the brain actually does), signal processing (light, sound, etc.) [2].

The human nervous system is very complex. According to various studies, the average human brain has tens, and sometimes hundreds of billions of neurons, and the number of connections between them is orders of magnitude higher. A unique feature of a neuron is its ability to transmit, receive, and process electrochemical signals transmitted by nerve connections. Groups of neurons and their connections form the communication system of the human brain.

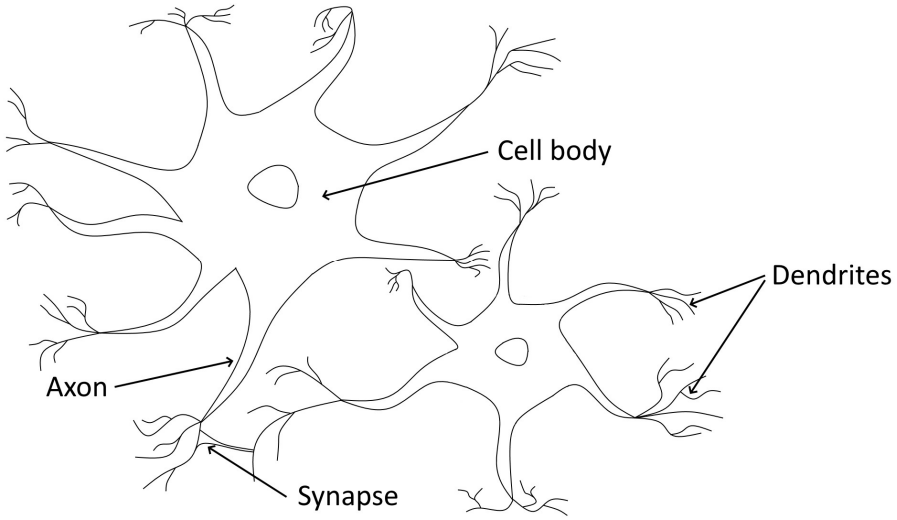


Figure 1.1 – The structure of a biological neuron

Figure 1.1 shows the structure of two biological neurons. In this structure, the cell body deals with data processing. The signal from the neuron body is propagated along axons and dendrites, and then through synapses (bridges between two neurons) to other neurons. It is important that the signal transmitted through the synapse can change (amplitude, frequency). The neuronal body processes input data by summing them (data coming from other neurons). Based on the intensities of the incoming signals, the body of the neuron makes a decision to transmit the signal further, while

the incoming signals can both contribute to the excitation of the cell and hinder it. Thus, further signal transmission occurs if the total excitation in the body of the neuron exceeds some minimum threshold. Of course, it should be understood that this is a very simplified view of the structure of a biological neuron, but most artificial neural networks use these simple properties [1].

1.1.2 Mathematical representation of artificial neural networks

An artificial neural network usually consists of an input layer of neurons (which receives some data), hidden layers (there can be as many or none at all; their number affects the complexity of the network), and an output layer of neurons.

The input layer of neurons can be thought of as sensors that receive some information. The hidden layers perform some transformations of the input data, and the output layer of the neurons is essentially the output of the network. Drawing analogies with neurobiology, we have some stimulus (input) and we want to get some reaction (output). Such an analogy is quite successful in solving some problems based on neural networks.

A typical architecture of a neural network, which is often used in various educational materials in the form of a graph, is a fully connected forward propagation network. This means that any signals from the input layer propagate in only one direction (from the input to the output of the network through the hidden layers), with each neuron of the next layer connected to the neurons of the previous one, and so on. The structure of such a network is shown in Figure 1.2. If one or more links of a fully connected direct propagation network are missing, such a network is called partially connected.

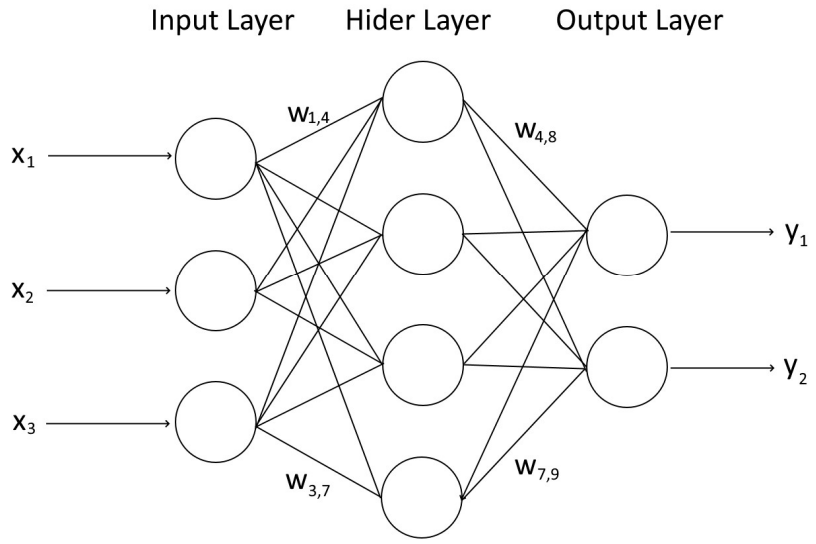


Figure 1.2 – The structure of a fully connected forward propagation neural network

Where x are the inputs of the network, $w_{i,j}$ are the connection weights of the i-th and j-th neurons, and y are the outputs of the network. Each neuron performs the summation of incoming signals (by analogy with the cell body in biological networks). The mathematical description of determining the output signal of a neuron is presented in formula 1.1 [1].

$$h_i = \sigma\left(\sum_{j=1}^{N} V_{ij} x_j\right) \quad (1.1)$$

The values of the neurons of the previous layer xi,j are multiplied by the weight coefficients wi,j and summed. The resulting value is called the weighted sum. The values of the weighting coefficients for each connection are different. The weighted sum is not normalized. For its normalization, the activation function σ (sum) is selected. The activation function that is mentioned most often when reviewing neural networks is the logistic function (or sigmoid, formula 1.2 [1]). Also, the hyperbolic tangent f(x) = th(x) can often be found. The task of the activation function is to bring the output values of the neurons into a range that can be used for transmission further along the neural network (usually this value is from -1 to 1 or from 0 to 1). By analogy with electronic systems, the activation function can be considered a nonlinear

amplifier.

$$f(x) = \frac{1}{1 + \exp(-\mathrm{x})} \quad (1.2)$$

The functional scheme (built using formula 1.1) of any neuron of a fully connected forward propagation network is presented in Figure 1.3.

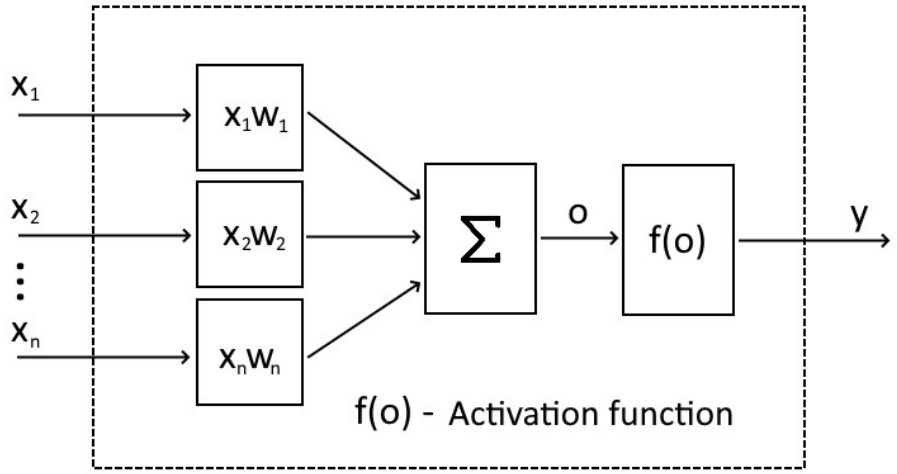


Figure 1.3 – Functional diagram of an artificial neuron

Thus, a neural network can be thought of as a very complex nonlinear function with complex inputs and outputs. This allows it to approximate any other function with an accuracy determined by its structure. At the same time, the inputs and outputs of the network can be encoded with any binary data (text, image pixel colors, sensor values, etc., etc.), which provides a very wide range of its application [3,4].

1.2. Analysis of the state of development of neural networks

1.2.1 Main areas of research

Research into the workings of the human brain inspired scientists to develop a mathematical model that would replicate its properties. However, in the course of solving this problem, other problems arose, which are related to the fact that the brain, in particular of a person, is a very complex structure that is difficult to describe mathematically. However, a better understanding of the functioning of neurons has made it possible to create different models to test their research. Modeling, even in a simplified form, made it possible to obtain interesting results. Properties of such models not only imitate some functions of the brain, but also have other features. Based on this, further research of neural networks can be divided into two directions, namely:

1. To understand how the human brain actually works on a physical and psychological level. To understand consciousness, etc.

2. On the basis of a clear model, develop new computer systems (artificial neural networks) that allow solving applied problems that are too complicated for traditional algorithms.

Actually, considering artificial neural networks, we are talking about the second direction of research [1].

Artificial neural networks have been a hot topic of research since around the 1980s to the present. We use only the properties of the brain in terms of information processing, the way in which it conducts them and abstract from the brain as a whole.

1.2.2 The formation of artificial neural networks as a separate branch of science

The study of neural networks began in the 1940s. American psychologist Warren McCulloch and mathematician Walter Pitts proposed the world's first M-P neural network model. It is very simple, but important for further development. In this model, the algorithm is implemented by considering the neuron as a functional logic device,

thus, a theoretical study of the artificial neural network model begins.

At this stage of the research, a problem arose with how to train this model. As mentioned earlier, a neural network can be represented as a very complex function. If its parameters (weighting coefficients for each of the connections) are chosen correctly, it is able to approximate another function, even a very complex one (this is an empirical observation, but it was discovered later). It is obvious that it is difficult to select these parameters manually for small structures, and for large neural networks (modern ones have at least thousands of connections) it is impossible at all. Therefore, an algorithm was needed that would allow training a neural network. The first learning model was proposed by Donald Gabb in 1949. He hypothesized that the learning process occurs at the synaptic interface between neurons, and the intensity of synaptic connections varies depending on the activity of neurons before and after the synapse. This principle is called Gebb's rule, which is still used today in the training of neural networks in an expanded form.

Later, Rosenblatt proposed a model called a "perceptron" based on the original M-P model (this is the functional diagram of the artificial neuron discussed earlier). This model includes the basic principles on which modern research on artificial neural networks is based, regardless of their complexity. This is a model of a neural network with continuous adjustment of weight coefficients. After training, it can achieve the goal of classifying and recognizing a certain mode of the input vector. Although it is relatively simple, it is the first true neural network. Rosenblatt proved that two-layer networks can classify input data and also suggested an important research direction for multilayer networks with the ability to train hidden (intermediate) layers of the network to be able to solve more complex problems. However, even such a simple neural network made it possible to solve the problems of weather prediction, analysis of electrocardiograms and artificial vision.

This became a very important step in the development of artificial intelligence, because during this study it was found that an artificial neural network, the structure of which corresponds to some extent to neurophysiology, is capable of learning and solving problems [5].

1.2.3 Problems of artificial neural networks

After the first significant results in the training of artificial neural networks, scientists believed for some time that it is enough to increase the complexity of the network (the number of neurons and hidden layers of the network) and artificial intelligence will be able to approach human intelligence. It quickly became clear that this was not enough.

Minsky and Papert, one of the founders of artificial intelligence, conducted a mathematical study of the functions and limitations of the perceptron. He failed to solve the problem of classification of two types of linear inseparable samples. An example is a simple linear sensor that cannot perform XOR ("exclusive or"). These conclusions, the author's persuasive arguments and taking into account previous disappointments in artificial neural networks significantly slowed down the development of artificial neural networks for tens of years.

Returning to the present, it is important to note that most of the problems mentioned have not actually disappeared, but have only become more complicated. The structure of the neural network and the method of its training remain relevant to solving the problem. The backpropagation algorithm is used very often in training neural networks. To work, he needs a training sample, an input-output pair. On its basis, the weight coefficients of the neural network are adjusted in such a way that the network error is smaller. The problem is that it is impossible to say with certainty whether the network will be able to learn in a finite amount of time and what network structure should be chosen for this. In addition, the already trained network may not be optimal because the basis of this algorithm is the gradient descent method, which can lead to the formation of local minima. In other words, the network configuration (values of weighting coefficients) may not be optimal.

If we look at machine learning today, none of the artificial neural networks in use today are universal. Each of them has a number of disadvantages and limitations that allow them to be used only for specific tasks [1,5].

1.3 Features of the functioning of neural networks

1.3.1 Validity of the result of artificial intelligence

An important characteristic of artificial neural networks is their reliability. For example, the human brain in some cases shows incredible indicators of structural reliability and the ability to self-organize and rebuild. But it is not about that, because in digital, the neural network is only a stream of bytes without a structure. We are talking about the result of the network in specific tasks.

There is some peculiarity in the work of almost any neural network. For example, consider a fully connected forward-propagation network with a non-linear activation function trained using a back-propagation algorithm. For example, this network solves the classic problem of recognizing handwritten numbers. As mentioned earlier, a neural network can approximate some complex function if some coefficients are chosen correctly. A trivial task that is often considered as an example is the recognition of handwritten digits. In this case, the function receives an image as input, and the output is the probability that this image is one of the numbers by which the network is trained. But since we are approximating the original function (actually it does not exist - it is a set of image-digit pairs), the result will be approximate. That is, at the output of the trained neural network, we will never get a 100% result for any of the numbers. In fact, this is not a problem, because it is important that the output neuron responsible for a specific number has the highest probability among all. The problem is that the neural network can make a mistake precisely at the stage of the result in working conditions (when it has no clue as to which particular number is in front of it and needs to give the result). Neural networks now have a very low error rate (98% or more correct answers). Such a low probability of error is important when neural networks can be applied where a person's life may depend on their decision.

The problem may actually arise precisely because we are making assumptions about the computer's inability to make mistakes. If the artificial neural network will sometimes make mistakes, then we conclude that it is unreliable. But in some problems that were previously considered unsolvable for a computer, neural networks already show results that are better than humans [1].

1.3.2 Main properties and advantages

Considering artificial neural networks and observing the results of their work, you can notice two important features:

1. Calculations that are necessary for the functioning of neural networks can be performed in parallel. Each layer of a neural network is a set of neurons for which it is necessary to perform weighted sum and normalization operations, which do not depend on other neurons of the same layer.

2. Neural networks are capable of self-learning and generalization. Generalization in this case is the ability to give the correct answer to input data that was not encountered during training. For example, a network that learned to recognize the presence of a face in an image should respond correctly if it is presented with an unfamiliar image.

Using these features, neural networks are able to solve large-scale problems that cannot be solved by standard algorithms. In practice, as mentioned earlier, universal neural networks do not exist. If the task is too complex, it is advisable to break it down into several simple tasks and train different networks to perform each of them. Or use combinations of neural networks and conventional algorithms. It is important to understand that neural networks give only an approximate result and standard algorithms can be much more effective in trivial tasks. The search for new problems where the application of neural networks is possible is a separate study.

If we compare artificial neural networks with standard algorithms, then their use provides the following useful properties to the system:

1. Nonlinearity. Artificial neurons can be both linear and non-linear, which in turn determines the activation function. Neural networks, the structure of which consists of non-linear neurons, acquires the property of non-linearity. Non-linearity of this type is somewhat special, because in this case it is distributed over the network. The property of non-linearity is important when the physical mechanism underlying the input signal is also non-linear (human language, images, video, etc.). In addition, the error backpropagation algorithm, which is often used in training neural networks,

uses a modified stochastic descent method, for which it is necessary to determine the derivative of the activation function. The activation function must be non-linear over the operating range.

2. Representation of output information through input (input-output mapping). Learning with a teacher (supervised learning) involves the presence of some sample of data, examples (training samples). An example is a pair of inputs and their corresponding outputs. Learning involves changing the weighting coefficients in such a way that the correct answer is obtained at the output of the network. Training continues until changes in weighting coefficients become insignificant.

3. Adaptivity. Neural networks have the ability to adapt their weights in such a way as to match the environment. Actually, a neural network trained to operate in a specific environment can easily be adapted to work in a new environment, if its parameters do not differ significantly. In addition, neural networks that learn in real time can be created to work in a nonstationary environment in which statistics change over time.

4. Obviousness of the answer (evidential response). Using the example of a classic sample classification problem, you can build a neural network in such a way that during training, at the output, it takes into account not only the specific class, but also how confident the network is in its answers, to increase the confidence of the decisions made by the network.

5. Contextual information. Knowledge about the objects submitted to the input of the neural network is stored in the weighting coefficients of individual neural connections. At the same time, in this paradigm, all neurons are interconnected and no arbitrary neuron without context carries useful information.

6. Fault tolerance. Neural networks presented in electronic form are potentially fault-tolerant. Failure of individual neurons or their connections leads to difficulty in obtaining reliable information. However, given the way information is stored in the neural network (decentralized and contextual), only significant damage to the structure of the neural network can lead to its inability to perform its work.

7. Scalability. The structure of a neural network is represented by simple

elements - neurons and their connections. The possibilities for scaling neural networks are limited only by the capabilities of computing technology and the adequacy of the approach to the given task.

8. Uniformity of analysis and design. Neural networks are a universal tool for processing data. The same project solution can be used to solve different problems. This is due to the fact that the basic principles of functioning of different types of neural networks are the same.

1.4 Classification of artificial neural networks

There are many many different configurations of neural networks. However, most of them can be classified according to some basic features. Actually, neural networks can be divided according to the topology of the connections between neurons and the type of artificial neurons used.

A neural network can be represented as a directed, weighted graph. In this way, the directions of signal transmission in the neural network and the weighting coefficients of each connection are traced.

According to the method of building connections, neural networks can be divided into feed forward propagation (in which the graph will not have loops) and neural networks with feedback [6].

1.4.1 Forward propagation neural networks

Single layer perceptron. In such a neural network model, all elements of the input layer are connected to the output layer through connections. It is the simplest network of direct distribution. [7].

Rosenblatt multilayer perceptron. This is a perceptron in which there is at least one hidden (associative) layer. Unlike a single-layer, this type of perceptron is able to solve the classic XOR problem.

Rumelhart multilayer perceptron. This type of neural network is a partial case of Rosenblatt's perceptron. The difference is that learning occurs using the method of backpropagation of the error. At the same time, all layers of the network are trained

[8].

Convolutional neural networks. This type of neural network uses the principles of human vision. When analyzing an image, we look at its individual parts with some filters. This operation is called convolution. The output of this network is a map of image features, which in classification tasks fed to the input of a multilayer perceptron, which already gives the probability of the image belonging to some classes [9].

Radial basis function network. As a rule, such a network includes three layers: an input, an output, and a hidden layer of neurons with a radial basis activation function. The same value is applied to the input neurons of this neural network. This network model allows you to efficiently perform function approximation tasks [10].

1.4.2 Neural networks with feedback

Kohonen's self-organizing maps. A feature of this type of networks is learning without a teacher. In addition, the principle of competitive learning is used instead of loss function optimization. That is, only the signal most similar to the incoming signal is transmitted - others are ignored. This approach allows the network to find regularities in input data that reinforce similar features [11].

Generative competitive network. An example of the operation of this type of network is the generation of human faces. In fact, this system consists of two neural networks. One neural network tries to generate a realistic image of a human face, while another tries to identify a fake one. This type of training is training without a teacher [12].

Hopfield neural network. A feature of this type of neural networks is the presence of a symmetrical matrix of connections. The operation of this network is reduced to some local minimum. This structure allows the network to perform the function of a filter and auto-associative memory, restoring damaged data on the basis of stored data [13].

1.5 Methods of using neural networks

Artificial neural networks have been finding new areas of application since the beginning of the active period of their research. The property of neural networks to approximate complex functions allows it to describe complex nonlinear processes.

Physical processes. Most physical processes are complex. Even calculating the flight path of a body taking into account many parameters can be difficult. But neural networks approximate complex functions well. Knowing some parameters of the atmosphere at different moments of time and the weather condition that corresponds to them, you can teach a neural network to predict weather based on new indicators.

Medicine. The ability of neural networks to self-organize makes it possible to identify patterns between diseases and patients. A trained network can diagnose a patient's disease based on previous experience.

Automated control systems. Management of production, and more specifically, individual production processes. A neural network can learn to change the system parameters based on some inputs about the state of production in an optimal way.

A classic task of pattern recognition or classification. Submit an image to the input of the network and receive at the output a probabilistic estimate of the network's belonging to this or that class. This also includes pattern recognition in real time [1,2].

Lossy data compression. The property of neural networks to find common features in various input data allows it to be used for a compact representation of the same data, for example, images [14].

ДОДАТОК Б

```
//Layer.compute
RWStructuredBuffer<float> Result;
RWStructuredBuffer<float> Deltas;
RWStructuredBuffer<float> inputs;
RWStructuredBuffer<float> outputDeltas;
int batchLength;

//Convolutional.compute
#pragma kernel ZeroPadding;
#pragma kernel FeedForward;
#pragma kernel BiasesDeltas;
#pragma kernel WeightsDeltas;
#pragma kernel InputDeltas;
static const int kernelSize = 3;
static const int padding = kernelSize / 2;
#include "Includes/Layer.compute"
RWStructuredBuffer<float> zpInputs;
RWStructuredBuffer<float> zpActivationsDeltas;
RWStructuredBuffer<float> weights;
RWStructuredBuffer<float> weightsDeltas;
RWStructuredBuffer<float> biases;
RWStructuredBuffer<float> biasesDeltas;
int filters;
int inputSize;
int paddingSize;
int inputChannels;
[numthreads(8, 8, 8)]
void ZeroPadding(int3 id : SV_DispatchThreadID)
{
	if (id.z >= inputSize * inputSize || id.y >= inputChannels || id.x >= batchLength) return;
	int i = id.z / inputSize, j = id.z % inputSize;
	zpInputs[paddingSize * (paddingSize * (inputChannels * id.x + id.y) + i + padding) + j + padding] =
inputs[inputSize * (inputSize * (inputChannels * id.x + id.y) + i) + j];
}
[numthreads(8, 8, 8)]
void FeedForward(int3 id : SV_DispatchThreadID)
{
	if (id.z >= inputSize * inputSize || id.x >= batchLength || id.y >= filters) return;
	int i = id.z / inputSize, j = id.z % inputSize;
```

```
		float activation = biases[id.y];
		for (int w1 = 0; w1 < kernelSize; w1++)
		for (int w2 = 0; w2 < kernelSize; w2++)
		for (int k = 0; k < inputChannels; k++)
			activation += weights[kernelSize * (kernelSize * (inputChannels * id.y + k) + w1) + w2] *
			zpInputs[paddingSize * (paddingSize * (inputChannels * id.x + k) + i + w1) + j + w2];
		Result[inputSize * (inputSize * (filters * id.x + id.y) + i) + j] = activation;
}
[numthreads(8, 8, 1)]
void BiasesDeltas(int3 id : SV_DispatchThreadID)
{
		if (id.x >= batchLength || id.y >= filters) return;
		float biasDelta = 0;
		for (int i = 0; i < inputSize; i++)
		for (int j = 0; j < inputSize; j++)
		{
			float value = outputDeltas[inputSize * (inputSize * (filters * id.x + id.y) + i) + j];
			zpActivationsDeltas[paddingSize * (paddingSize * (filters * id.x + id.y) + i + padding) + j + padding]
= value;
			biasDelta += value;
		}
		biasesDeltas[id.y] = biasDelta;
}
[numthreads(8, 8, 8)]
void WeightsDeltas(int3 id : SV_DispatchThreadID)
{
		if (id.z >= kernelSize * kernelSize || id.y >= filters * inputChannels || id.x >= batchLength) return;
		int r = id.y / inputChannels, k = id.y % inputChannels, w1 = id.z / kernelSize, w2 = id.z % kernelSize;
		float weightDelta = 0;
		for (int i = 0; i < inputSize; i++)
		for (int j = 0; j < inputSize; j++)
			weightDelta += outputDeltas[inputSize * (inputSize * (filters * id.x + r) + i) + j] *
			zpInputs[paddingSize * (paddingSize * (inputChannels * id.x + k) + i + w1) + j + w2];
		weightsDeltas[kernelSize * (kernelSize * (inputChannels * (filters * id.x + r) + k) + w1) + w2] = weightDelta;
}
[numthreads(8, 8, 8)]
void InputDeltas(int3 id : SV_DispatchThreadID)
{
		if (id.z >= inputSize * inputSize || id.y >= inputChannels || id.x >= batchLength) return;
		int i = id.z / inputSize, j = id.z % inputSize;
		float inputDelta = 0;
		for (int r = 0; r < filters; r++)
```

```
        for (int w1 = 0; w1 < kernelSize; w1++)
        for (int w2 = 0; w2 < kernelSize; w2++)
                inputDelta += weights[kernelSize * (kernelSize * (inputChannels * r + id.y) + kernelSize - 1 - w1) +
kernelSize - 1 - w2] *
                zpActivationsDeltas[paddingSize * (paddingSize * (filters * id.x + r) + i + w1) + j + w2];
        Deltas[inputSize * (inputSize * (inputChannels * id.x + id.y) + i) + j] = inputDelta;
}

//BatchNorm.compute
#pragma kernel Inference;
#pragma kernel MuSigma;
#pragma kernel FeedForward;
#pragma kernel Momentum;
#pragma kernel ParamDeltas;
#pragma kernel MuSigmaDeltas;
#pragma kernel InputDeltas;
#include "Includes/Layer.compute"
RWStructuredBuffer<float> mu, sigma, mu0, sigma0, muDeltas, sigmaDeltas;
RWStructuredBuffer<float> normalizations, normalizationsDeltas;
RWStructuredBuffer<float> betas, betasDeltas, betasHistory;
RWStructuredBuffer<float> gammas, gammasDeltas, gammasHistory;
int depth, width, height;
float epsilon, momentum;
[numthreads(8, 8, 8)]
void Inference(int3 id : SV_DispatchThreadID)
{
        if (id.x >= depth || id.z >= width * height || id.y >= batchLength) return;
        float _mu = mu0[id.x], _sigma = sigma0[id.x], beta = betas[id.x], gamma = gammas[id.x];
        int idx = width * (height * (depth * id.y + id.x) + id.z / height) + id.z % height;
        float normalize = (inputs[idx] - _mu) / sqrt(_sigma + epsilon);
        normalizations[idx] = normalize;
        Result[idx] = normalize * gamma + beta;
}
[numthreads(8, 1, 1)]
void MuSigma(int3 id : SV_DispatchThreadID)
{
        if (id.x >= depth) return;
   float _mu = 0, _sigma = 0;
   for (int i = 0; i < width; i++)
   for (int j = 0; j < height; j++)
   for (int b = 0; b < batchLength; b++)
      _mu += inputs[width * (height * (depth * b + id.x) + i) + j];
```

```
    mu[id.x] = (_mu /= (width * height * batchLength));
    for (int i = 0; i < width; i++)
    for (int j = 0; j < height; j++)
    for (int b = 0; b < batchLength; b++)
        _sigma += pow(inputs[width * (height * (depth * b + id.x) + i) + j] - _mu, 2);
    sigma[id.x] = (_sigma /= (width * height * batchLength));
}
[numthreads(8, 1, 1)]
void FeedForward(int3 id : SV_DispatchThreadID)
{
        if (id.x >= depth || id.z >= width * height || id.y >= batchLength) return;
        float _mu = mu[id.x], _sigma = sigma[id.x], beta = betas[id.x], gamma = gammas[id.x];
        int idx = width * (height * (depth * id.y + id.x) + id.z / height) + id.z % height;
        float normalize = (inputs[idx] - _mu) / sqrt(_sigma + epsilon);
        normalizations[idx] = normalize;
        Result[idx] = normalize * gamma + beta;
}
[numthreads(8, 1, 1)]
void Momentum(int3 id : SV_DispatchThreadID)
{
        if (id.x >= depth) return;
    mu0[id.x] = mu0[id.x] * momentum + mu[id.x] * (1 - momentum);
    sigma0[id.x] = sigma0[id.x] * momentum + sigma[id.x] * (1 - momentum);
}
[numthreads(8, 1, 1)]
void ParamDeltas(int3 id : SV_DispatchThreadID)
{
        if (id.x >= depth) return;
    float gamma = gammas[id.x], betaDelta = 0, gammaDelta = 0;
    for (int i = 0; i < width; i++)
    for (int j = 0; j < height; j++)
    for (int b = 0; b < batchLength; b++)
    {
        int idx = width * (height * (depth * b + id.x) + i) + j;
        float value = outputDeltas[idx];
        normalizationsDeltas[idx] = value * gamma;
        gammaDelta += value * normalizations[idx];
        betaDelta += value;
    }
    gammasDeltas[id.x] = (gammaDelta /= (width * height * batchLength));
    betasDeltas[id.x] = (betaDelta /= (width * height * batchLength));
}
```

```
[numthreads(8, 1, 1)]
void MuSigmaDeltas(int3 id : SV_DispatchThreadID)
{
   if (id.x >= depth) return;
   float _mu = mu[id.x], _sigma = sigma[id.x], firstSum = 0, secondSum = 0, thirdSum = 0;
   for (int i = 0; i < width; i++)
   for (int j = 0; j < height; j++)
   for (int b = 0; b < batchLength; b++)
   {
      int idx = width * (height * (depth * b + id.x) + i) + j;
      float norm = normalizationsDeltas[idx], sub = inputs[idx] - _mu;
      firstSum += norm * sub;
      secondSum += norm;
      thirdSum += sub;
   }
   sigmaDeltas[id.x] = firstSum * -0.5f * pow(_sigma + epsilon, -1.5f);
   muDeltas[id.x] = secondSum / -sqrt(_sigma + epsilon) + sigmaDeltas[id.x] * -2 * thirdSum / (width * height *
batchLength);
}
[numthreads(8, 8, 8)]
void InputDeltas(int3 id : SV_DispatchThreadID)
{
         if (id.x >= depth || id.z >= width * height || id.y >= batchLength) return;
         int idx = width * (height * (depth * id.y + id.x) + id.z / height) + id.z % height;
   float _mu = mu[id.x], _sigma = sigma[id.x];
         Deltas[idx] = normalizationsDeltas[idx] / sqrt(_sigma + epsilon) + (sigmaDeltas[id.x] * 2 * (inputs[idx] - _mu)
+ muDeltas[id.x]) / (width * height * batchLength);
}

//Dense.compute
#pragma kernel FeedForward
#pragma kernel BiasesDeltas
#pragma kernel InputAndWeightDeltas
#include "Includes/Layer.compute"
RWStructuredBuffer<float> weightsDeltas;
RWStructuredBuffer<float> biasesDeltas;
RWStructuredBuffer<float> weights;
RWStructuredBuffer<float> biases;
int size;
int inputSize;
[numthreads(8, 8, 1)]
void FeedForward(int3 id : SV_DispatchThreadID)
```

```
{
	if (id.y >= size || id.x >= batchLength)  return;
	float activation = biases[id.y];
	for (int j = 0; j < inputSize; j++)
		activation += inputs[id.x * inputSize + j] * weights[id.y * inputSize + j];
	Result[id.x * size + id.y] = activation;
}
[numthreads(8, 8, 1)]
void BiasesDeltas(int3 id : SV_DispatchThreadID)
{
	if (id.y >= size || id.x >= batchLength)  return;
	biasesDeltas[id.x * size + id.y] = outputDeltas[id.x * size + id.y];
}
[numthreads(8, 8, 1)]
void InputAndWeightDeltas(int3 id : SV_DispatchThreadID)
{
	if (id.y >= inputSize || id.x >= batchLength)  return;
	float inputDelta = 0, input = inputs[id.x * inputSize + id.y];
	for (int i = 0; i < size; i++)
	{
		float delta = outputDeltas[id.x * size + i];
		inputDelta += weights[i * inputSize + id.y] * delta;
		weightsDeltas[inputSize * (size * id.x + i) + id.y] = input * delta;
	}
	Deltas[id.x * inputSize + id.y] = inputDelta;
}

//ReLU.compute
#pragma kernel FeedForward
#pragma kernel BackPropagation
#include "Includes/Layer.compute"
int FanOut;
[numthreads(8, 8, 1)]
void FeedForward(int3 id : SV_DispatchThreadID)
{
	if (id.x >= batchLength || id.y >= FanOut) return;
	float value = inputs[id.x * FanOut + id.y];
	Result[id.x * FanOut + id.y] = value > 0 ? value : 0;
}
[numthreads(8, 8, 1)]
void BackPropagation(int3 id : SV_DispatchThreadID)
{
```

```
	if (id.x >= batchLength || id.y >= FanOut) return;
	Deltas[id.x * FanOut + id.y] = Result[id.x * FanOut + id.y] > 0 ? outputDeltas[id.x * FanOut + id.y] : 0;
}

//MaxPooling.compute
#pragma kernel FeedForward
#pragma kernel BackPropagation
#include "Includes/Layer.compute"
RWStructuredBuffer<int> cachedIdx;
int kernelSize;
int inputSize;
int poolSize;
int inputChannels;
[numthreads(8, 8, 8)]
void FeedForward(int3 id : SV_DispatchThreadID)
{
	if (id.z >= poolSize * poolSize || id.x >= batchLength || id.y >= inputChannels) return;
	int i = id.z / poolSize, j = id.z % poolSize, maxIdx;
	float max = 0;
	for (int w1 = 0; w1 < kernelSize; w1++)
	for (int w2 = 0; w2 < kernelSize; w2++)
	{
		int idx = inputSize * (inputSize * (inputChannels * id.x + id.y) + i * kernelSize + w1) + j * kernelSize
+ w2;
		float tmp = inputs[idx];
		if (tmp >= max)
		{
			maxIdx = idx;
			max = tmp;
		}
	}
	int pool_idx = poolSize * (poolSize * (inputChannels * id.x + id.y) + i) + j;
	cachedIdx[pool_idx] = maxIdx;
	Result[pool_idx] = max;
}
[numthreads(8, 8, 8)]
void BackPropagation(int3 id : SV_DispatchThreadID)
{
	if (id.z >= poolSize * poolSize || id.x >= batchLength || id.y >= inputChannels) return;
	int i = id.z / poolSize, j = id.z % poolSize, pool_idx = poolSize * (poolSize * (inputChannels * id.x + id.y) + i)
+ j;
	int maxIdx = cachedIdx[pool_idx];
```

```
            float delta = outputDeltas[pool_idx];
            for (int w1 = 0; w1 < kernelSize; w1++)
            for (int w2 = 0; w2 < kernelSize; w2++)
            {
                        int idx = inputSize * (inputSize * (inputChannels * id.x + id.y) + i * kernelSize + w1) + j * kernelSize
+ w2;
                        Deltas[idx] = idx == maxIdx ? delta : 0;
            }
}

//UpSampling.compute
#pragma kernel FeedForward
#pragma kernel BackPropagation
#include "Includes/Layer.compute"
int size, inputSize, inputChannels, scale;
[numthreads(8,8,1)]
void FeedForward(uint3 id : SV_DispatchThreadID)
{
   if (id.x >= batchLength || id.y >= inputChannels || id.z >= inputSize) return;
   for (int j = 0; j < inputSize; j++)
   {
      float input = inputs[inputSize * (inputSize * (inputChannels * id.x + id.y) + id.z) + j];
      for (int w1 = 0; w1 < scale; w1++)
      for (int w2 = 0; w2 < scale; w2++)
         Result[size * (size * (inputChannels * id.x + id.y) + id.z * scale + w1) + j * scale + w2] = input;
   }
}
[numthreads(8, 8, 1)]
void BackPropagation(uint3 id : SV_DispatchThreadID)
{
            if (id.x >= batchLength || id.y >= inputChannels || id.z >= inputSize) return;
            for (int j = 0; j < inputSize; j++) {
                        float value = 0;
                        for (int w1 = 0; w1 < scale; w1++)
                        for (int w2 = 0; w2 < scale; w2++)
                                    value += outputDeltas[size * (size * (inputChannels * id.x + id.y) + id.z * scale + w1) + j *
scale + w2];
                        Deltas[inputSize * (inputSize * (inputChannels * id.x + id.y) + id.z) + j] = value;
            }
}

//LossFunction.compute
```

```
RWStructuredBuffer<float> outputs;
RWStructuredBuffer<float> targets;
RWStructuredBuffer<float> errors;
RWStructuredBuffer<float> deltas;
int FanOut;
int batchLength;
```

//SoftMaxCrossEntropy.compute

```
#pragma kernel Calculate
#include "Includes/LossFunction.compute"
RWStructuredBuffer<float> predictedClasses;
[numthreads(8, 1, 1)]
void Calculate(int3 id : SV_DispatchThreadID)
{
	if (id.x >= batchLength) return;
	float sum = 0, error = 0;
	for (int i = 0; i < FanOut; i++)
	{
		float value = exp(outputs[id.x * FanOut + i]);
		outputs[id.x * FanOut + i] = value;
		sum += value;
	}
	float targetValue = 0, maxValue = 0;
	for (int i = 0; i < FanOut; i++)
	{
		float value = outputs[id.x * FanOut + i] / sum, target = targets[id.x * FanOut
+ i];
		deltas[id.x * FanOut + i] = value - target;
		error -= target * log(clamp(value, 0, value) + 0.000000001f);
		if (target != 0) targetValue = value;
		maxValue = max(maxValue, value);
		outputs[id.x * FanOut + i] = value;
	}
	predictedClasses[id.x] = targetValue == maxValue ? 1 : 0;
	errors[id.x] = error;
}
```

//MSE.compute

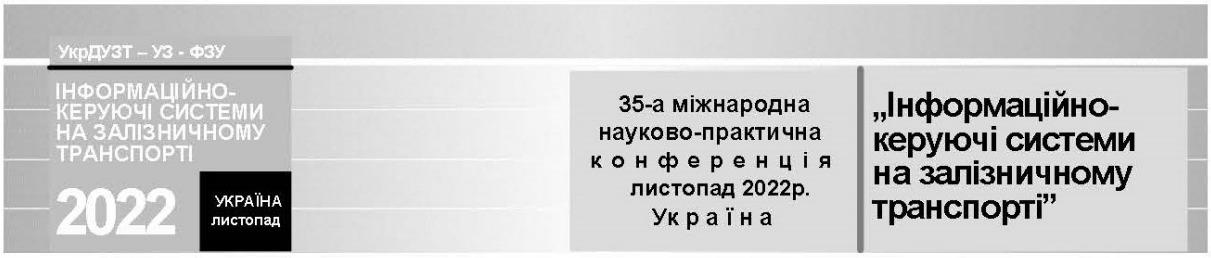
```
#pragma kernel Calculate
#include "Includes/LossFunction.compute"
[numthreads(8, 1, 1)]
void Calculate(int3 id : SV_DispatchThreadID)
{
	if (id.x >= batchLength) return;
	float error = 0;
```

```
        for (int i = 0; i < FanOut; i++)
        {
                int idx = id.x * FanOut + i;
                float value = outputs[idx] - targets[idx];
                error += value * value;
                deltas[idx] = value;
        }
        errors[id.x] = error;
}


//Optimizer.compute
static const int historySize = 4;
int batchLength, paramLength;
RWStructuredBuffer<float> values;
RWStructuredBuffer<float> deltas;
RWStructuredBuffer<float> history;
float learningRate, gradClip, weightDecay;
float deltasFromBatch(int idx)
{
        float delta = 0;
        for (int b = 0; b < batchLength; b++)
                delta += deltas[b * paramLength + idx];
        if (gradClip > 0)  delta = clamp(delta, -gradClip, gradClip);
        if (weightDecay > 0) delta += weightDecay * values[idx];
        return delta / batchLength;
}

//SGD.compute
#pragma kernel Optimize
#include "Includes/Optimizer.compute"
float momentum;
bool nesterov;
[numthreads(32, 1, 1)]
void Optimize(int3 id : SV_DispatchThreadID)
{
        if (id.x >= paramLength) return;
        float delta = deltasFromBatch(id.x);
        int historyOffset = id.x * historySize;
        float first = momentum * history[historyOffset] + delta;
        history[historyOffset] = first;
        if (nesterov) delta += momentum * first;
```

```
        else delta = first;
        values[id.x] -= learningRate * delta;
}

//Adam.compute
#pragma kernel Optimize
#include "Includes/Optimizer.compute"
float epsilon;
float beta1;
float beta2;
float expbeta1;
float expbeta2;
bool amsgrad;
[numthreads(32, 1, 1)]
void Optimize(int3 id : SV_DispatchThreadID)
{
        if (id.x >= paramLength)  return;
        float delta = deltasFromBatch(id.x);
        int historyOffset = id.x * historySize;
        float first = beta1 * history[historyOffset] + (1 - beta1) * delta;
        float second = beta2 * history[historyOffset + 1] + (1 - beta2) * delta * delta;
        history[historyOffset] = first;
        history[historyOffset + 1] = second;
        float _first = first / (1 - expbeta1);
        float _second = second / (1 - expbeta2);
        if (amsgrad) {
                _second = max(_second, history[historyOffset + 2]);
                history[historyOffset + 2] = _second;
        }
        values[id.x] -= learningRate * _first / (sqrt(_second) + epsilon);
}
```

ДОДАТОК В

Міністерство освіти і науки України
Акціонерне товариство „Українська залізниця”
Транспортна академія України
Федерація залізничників України
Український державний університет залізничного транспорту



Оргкомітет:

Голова:

Панченко С. В., д.т.н., ректор Українського державного університету залізничного транспорту

Члени оргкомітету: Бабаєв М. М., д.т.н. (Україна), Бунчуков О. А. (Україна), Бутько Т. В., д.т.н. (Україна), Гаврилюк В. І., д.ф-м.н. (Україна), Гончаренко В. І. (Україна), Доценко С. І ., д.т.н. (Україна), Жуковицький І. В., д.т.н. (Україна), Каргін А. О., д.т.н. (Україна), Климаш М. М., д.т.н. (Україна), Збігнев Лукасік, д.т.н. (Польща), Марек Мезитис, д.т.н. (Латвія), Мойсеєнко В. І ., д.т.н. (Україна), Приходько С. І., д.т.н. (Україна), Рубан І. В., д.т.н. (Україна), Самсонкін В. М., д.т.н. (Україна), Серков О. А., д.т.н. (Україна), Скалозуб В. В., д.т.н. (Україна), Терещенко Ю. М. (Україна), Трубчанінова К. А., д.т.н. (Україна), Тьєрі Хорсін (Франція), Шиш В. О., к.т.н. (Україна), Штомпель М. А., д.т.н. (Україна)

2022 р.
11 листопада

м. Харків,
Україна

ТЕЗИ СТЕНДОВИХ ДОПОВІДЕЙ ТА ВИСТУПІВ УЧАСНИКІВ КОНФЕРЕНЦІЇ

HIGHLIGHTS OF REPORTS AND PRESENTATIONS OF PARTICIPANTS TO THE CONFERENCE

складає 4,89 хв., на станції Чоп – 25,34 хв., а на станції Мостиська-2 – 50,8 хв., при чому для поїздів за номерами 54/36 Перемишль – Київ, Одеса та 706 (Інтерсіті +) Перемишль – Київ вона була щоденною. Такий великий рівень затримок також впливає і на розклад руху інших поїздів, особливо це стосується поїздів 38 Ужгород – Одеса та 96 Рахів – Київ, які зі Львова до Києва та Одеси прямують як двогрупний поїзд і мусять чекати поїзд 54/36, що запізнюється. Наслідком цього є незадоволення пасажирів та велика кількість претензій до Акціонерного товариства «Українська залізниця». Основною причиною затримок (близько 80%) є оформлення та перевірка документів, як на українському, так і на польському кордоні. Всього через пункт пропуску Мостиська-2 проходить п'ять пар пасажирських поїздів, дві з яких мають категорію Інтерсіті +. Враховуючи це, а також покращення дружніх стосунків між Україною і Польщею, авторами даної публікації було запропоновано створення сумісного митного та прикордонного контролю між польськими та українськими прикордонниками на станції Мостиська-2 згідно з стандартами переглянутою Кіотської Конвенції [2].

Практика застосування сумісних кордонів вже давно використовується у світі, наприклад, між Німеччиною та Польщею (до вступу останньої до Європейського Союзу), США та Канадою та ін. [3]. Таке рішення дозволить значно зменшити затримки та час слідування пасажирських поїздів, покращити якість обслуговування пасажирів та збільшити економічний ефект від міжнародних пасажирських залізничних перевезень.

Список використаних джерел

1. У Радбезі ООН засудили російські фільтраційні табори та назвали кількість українських біженців [Електронний ресурс]. – URL: https://suspilne.media/279443-u-radbezi-oon-zasudili-rosijski-filtracijni-tabori-ta-nazvali-kilkist-ukrainskih-bizenciv/ /. – Дата звернення: 23.10.2022.
2. Міжнародна конвенція про спрощення і гармонізацію митних процедур (Кіотська конвенція) [Електронний ресурс]. – URL: https://zakon.rada.gov.ua/laws/show/995_643#Text – Дата звернення: 24.10.2022
3. Балака Є. І., Зоріна О. І., Колеснікова Н. М. та ін. Тенденції розвитку залізничних перевезень в провідних країнах світу. Залізничний транспорт України. 2000. №1. С. 22–23.

Шефер О. В., д.т.н., професор,
Фенько В. В., магістрант
(Національний університет «Полтавська політехніка імені Юрія Кондратюка»)

УДК 004.8

РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ДОСЛІДЖЕННЯ МЕТОДІВ ОБРОБКИ ІНФОРМАЦІЇ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

У сучасному світі дуже часто згадуються такі поняття як нейронні мережі, машинне навчання та машинній зір. Розвиток якісних та кількісних характеристик обчислювальної техніки та удосконалення методів побудови та навчання нейронних мереж дозволяють вирішувати задачі, які вважаються надскладними для звичайних алгоритмів. Яскравим прикладом є задачі машинного зору, до яких входить класифікація, локалізація, виділення контуру графічних образів і багато іншого. Крім того, зараз досліджуються випадки коли властивості нейронних мереж можна застосовувати на заміну звичайних алгоритмів, наприклад для стискання графічних образів [1].

Більшість нейронних мереж у своїй роботі опирається на одні й ті самі базові принципи, проте вибір оптимальних гіперпараметрів (кількість, параметри та способи організації зв'язків структурних блоків мережі, функції активації та втрат, метод оптимізації функції втрат та інше) може сильно відрізнятися в залежності від задачі, що вирішується. Таким чином, розв'язання кожної окремої задачі зводиться до емпіричного дослідження з вибору оптимальної архітектури мережі та інших параметрів для її ефективного навчання [2].

Розроблення програмного забезпечення із графічним інтерфейсом, що дозволяє виводити статистику навчання мережі (значення параметрів при навчання мережі, візуалізація активацій виходів згорткових шарів та інші параметри, що дозволяють відстежувати поведінку роботи мережі), може значно спростити вибір оптимальних початкових гіперпараметрів мережі для її подальшого навчання.

Окрім вищеперерахованих функцій дане програмне забезпечення, для ефективного навчання нейронних мереж у задачах машинного зору, повинно включати наступні структурні блоки:

- Система вводу/виводу із відповідним графічним інтерфейсом для завантаження навчальних даних (наприклад, промаркованих зображень для задач класифікації) та збереження/завантаження налаштувань мережі для її використання у подальшому;
- Можливість автоматичного збереження

параметрів мережі під час навчання (на кожній ітерації) для того щоб повернутися до оптимальних параметрів у разі помилки;

- Інтерфейс створення топології мережі та налаштування параметрів її окремих структурних блоків.

- Програмна реалізація усіх необхідних для роботи та навчання мережі алгоритмів.

За основу для такого програмного забезпечення можна брати готовий фрейворк для глибокого машинного навчання або реалізувати усю логіку роботи мережі самостійно у рамках конкретного проєкту (якщо є необхідність). Наявність такого програмного забезпечення у відкритому доступі дозволить редагувати окремі структурні блоки, що забезпечує гнучкість.

Загальна ідея розробки такого програмного забезпечення полягає в тому, що графічний інтерфейс значно простіший для сприйняття і забезпечує кращі можливості з повторного використання, адже окремі етапи побудови та налаштування моделі мережі можна спростити (сховати за інтерфейс).

Таким чином, розроблене програмне забезпечення дозволяє зменшити час на побудову та вибір параметрів для навчання моделі мережі. Подальші дослідження спрямовані на вирішення задач машинного зору (класифікації, стискання графічних образів) в рамках даного програмного забезпечення.

Список використаних джерел

1 Carsten Steger. Machine Vision Algorithms and Applications / Carsten Steger, Markus Ulrich, Christian Wiedemann., 2018. – 516 с.

2 Guanghui Lan. First-order and Stochastic Optimization Methods for Machine Learning / Guanghui Lan., 2020. – 582 с.

Жученко О. С., к.т.н., доцент (УкрДУЗТ),
Карпук В. Ю., магістрант (Національний університет «Полтавська політехніка імені Юрія Кондратюка»)

УДК 621.39

РОЗРОБКА ПРОТОКОЛУ ТЕЛЕМЕТРІЇ ДЛЯ ДИСТАНЦІЙНО-КЕРОВАНОЇ ТЕХНІКИ ТА ЙОГО ПРОГРАМНОЇ РЕАЛІЗАЦІЇ В УМОВАХ ЄВРОІНТЕГРАЦІЇ

У сучасному світі є популярним використання дистанційно-керованої техніки такої як: обладнання розумного будинку чи безпілотні літальні апарати (БПЛА). Для можливості їх роботи потрібні протоколи прикладного рівня, що зможуть описати правила обміну інформацією між ними та між людиною, яка є їх оператором. Наявність такої потреби підтверджується існуванням таких протоколів для дистанційно-керованої техніки як ModBus, UAVCan, UranusLink, MavLink, MQTT, CoAP тощо [1]. Однак, ці протоколи мають недоліки, які за певних умов можуть перешкоджати їх безпосередньому застосуванню, наприклад ModBus, UranusLink, MavLink передають дані у відритому форматі без можливості шифрування, MQTT та CoAP не мають можливості потокової передачі відео, MavLink маючи багатий функціонал є доволі складним для його імплементації розробником.

Альтернативним рішенням є розробка та реалізація власного протоколу для телеметрії та дистанційного управління, який може передавати потокові аудіо та відео дані, легкість реалізації для розробника та надає додаткову можливість шифрування трафіку.

Протокол є прикладним та призначений для обміну даними між керуючим оператором та дистанційно-керованою технікою. Цей протокол може використовуватися для віддаленого управління, моніторингу чи налаштування різної техніки.

Протокол працює поверх транспортного рівня тому він займає місце 7-го рівня у мережевій моделі OSI або 4-й прикладний рівень у моделі TCP/IP [2]. За рахунок цього він може бути переданий по будь-якому середовищу передачі.

Однією із головних особливостей є легкість його реалізації та використання для розробника. Зазначимо основні особливості протоколу телеметрії:

- протокол може передавати різні типи даних: команди керування, телеметричні дані, аудіо і відео потоки та спеціальні службові повідомлення;

- в залежності від типу даних, що передаються, протокол має різні механізми гарантії їх доставки та перевірки цілісності;

- у протоколі є необов'язкова можливість шифрування трафіку;

- більшість типів повідомлень мають представлення у двійковому вигляді, але частина має текстовий формат. Такий підхід забезпечує гнучкість розробки та невелике використання надлишкових даних;

- протокол може визначати пріоритет пакетам передачі. Такі пакети будуть оброблятися швидше.

Для розробки протоколу телеметрії створено декілька можливих типів повідомлень (рис. 1), керуючись загальноприйнятими рекомендаціями розробників протоколів [3]. Особливу увагу приділялося зменшенню розміру пакету із збереженням функціональності та гнучкості розробки для подальшої можливої модернізації.

Національний університет «Полтавська політехніка імені Юрія Кондратюка»
Навчально-науковий інститут інформаційних технологій і робототехніки
Кафедра автоматики, електроніки та телекомунікацій

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

НА ТЕМУ **РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ДОСЛІДЖЕННЯ МЕТОДІВ ОБРОБКИДАНИХ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ**

Виконав: студент групи 601 -ТТ Фенько В.В
Керівник кваліфікаційної роботи: Косенко В.В.

Полтава 2022

ТЕМА: РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ДОСЛІДЖЕННЯ МЕТОДІВ ОБРОБКИ ДАНИХ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

Мета: розробка програмного забезпечення із графічним інтерфейсом для спрощення аналізу навчання нейронних мереж

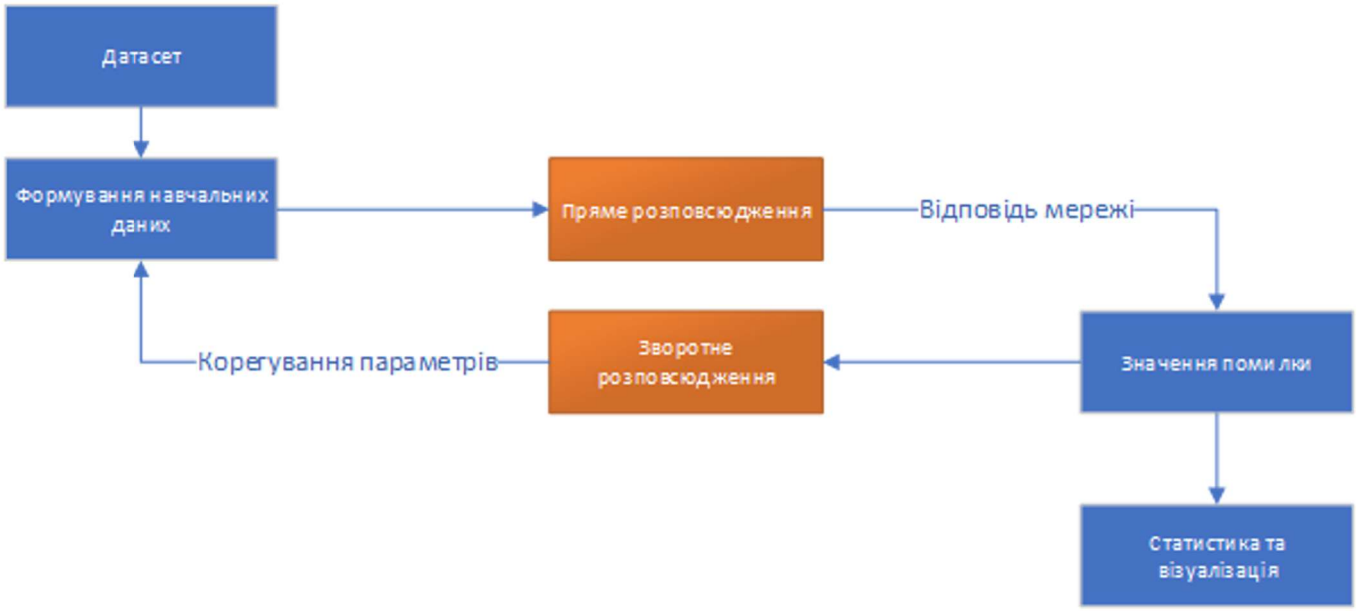
Об’єкт дослідження: методи обробки інформації на основі штучного інтелекту

Предмет дослідження: машинний зір у задачах класифікації та стиснення графічних образів

Задачі даної кваліфікаційної роботи:

- Аналіз сучасних тенденцій розвитку штучного інтелекту;
- Дослідження можливостей застосування нейронних мереж для обробки інформації;
- Вибір засобів для реалізації обраних напрямків дослідження;
- Розроблення програмного забезпечення із графічним інтерфейсом для навчання нейронних мереж;
- Вибір архітектури, навчання та дослідження мережі. Збір статистики.

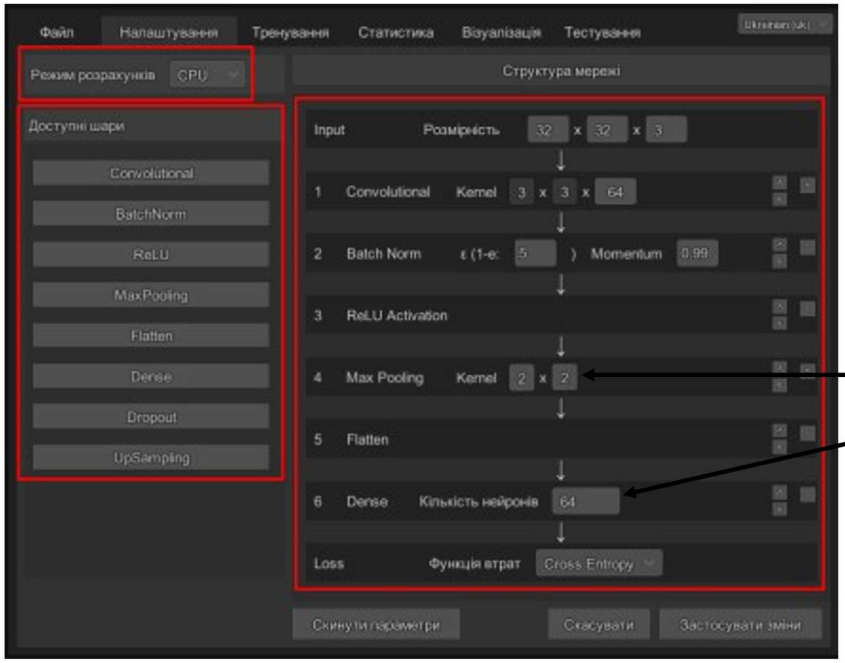
ПРОЦЕС СТВОРЕННЯ МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ



НАЛАШТУВАННЯ ПАРАМЕТРІВ МЕРЕЖІ

CPU/GPU

Перелік доступних структурних елементів



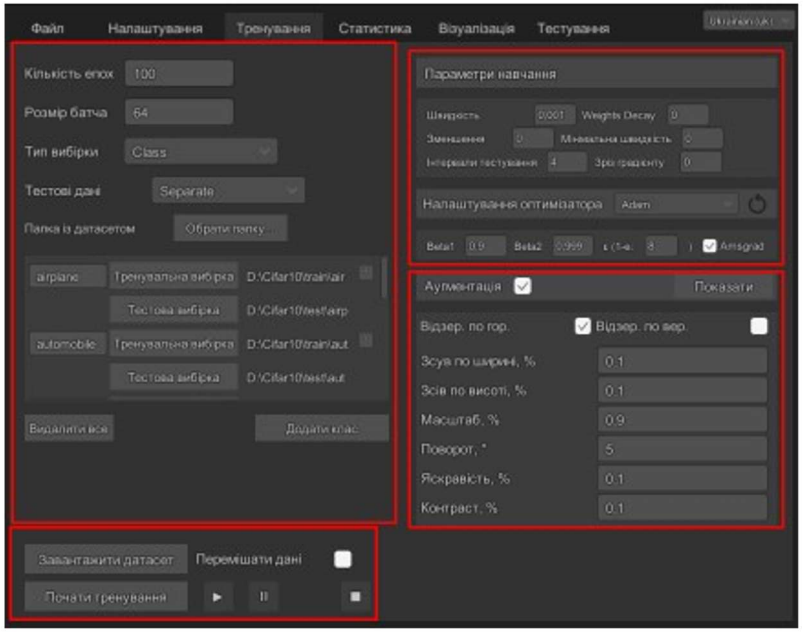
Топологія мережі

Гіперпараметри

КЕРУВАННЯ НАВЧАННЯМ МОДЕЛІ

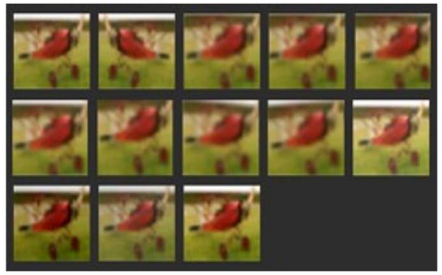
Налаштування датасету

Блок керування



Налаштування оптимізатора

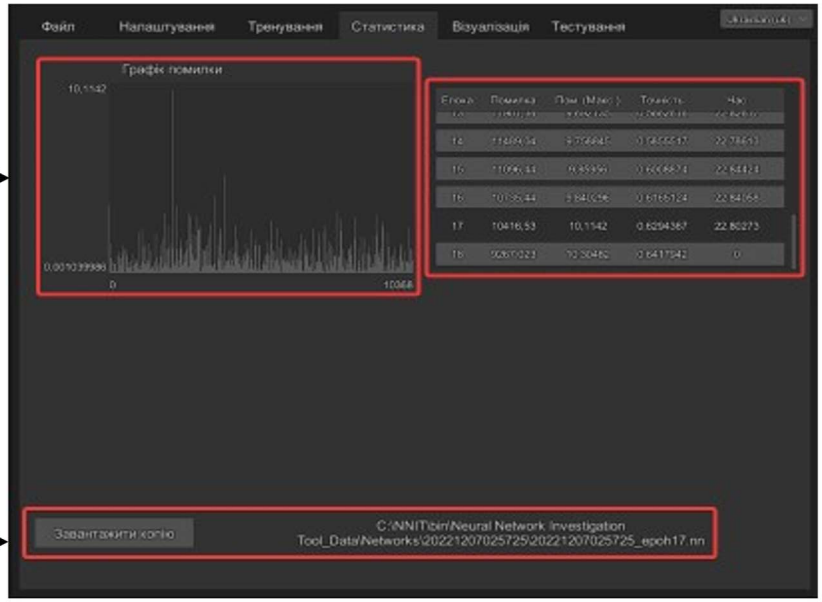
Аугментація вхідних даних



Результат аугментації

СТАТИСТИКА НАВЧАННЯ

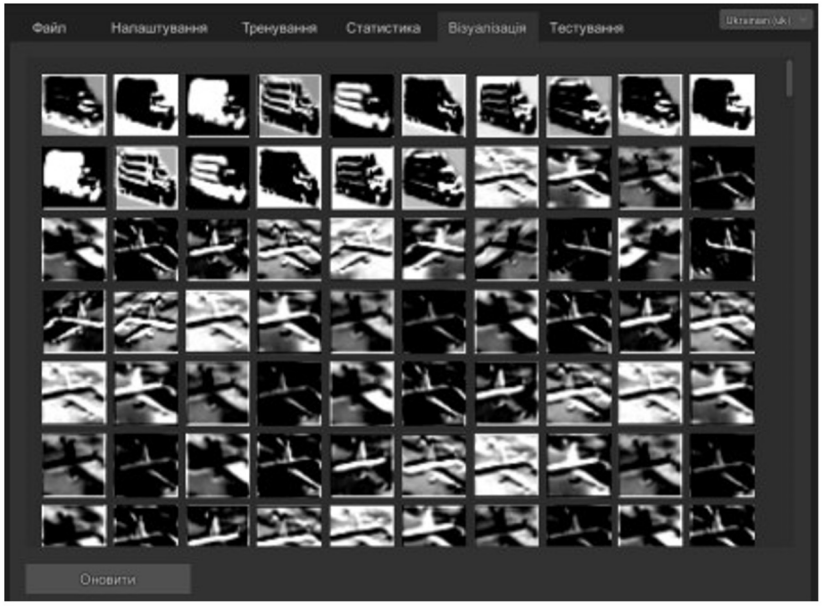
Графік помилки
для обраної епохи



Список-таблиця епох
навчання

BackUp обраної епохи

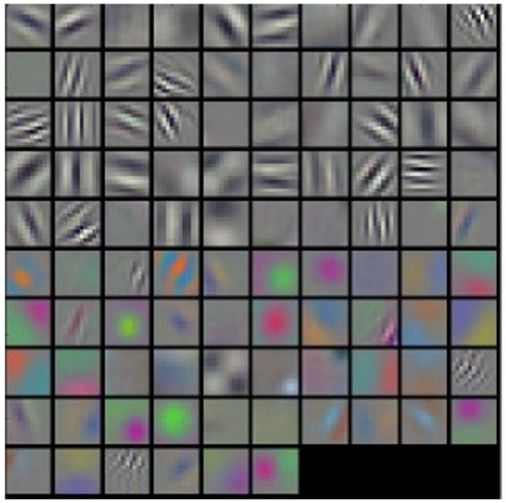
ВІЗУАЛІЗАЦІЯ РОБОТИ МЕРЕЖІ



Мапи ознак (Feature Maps)

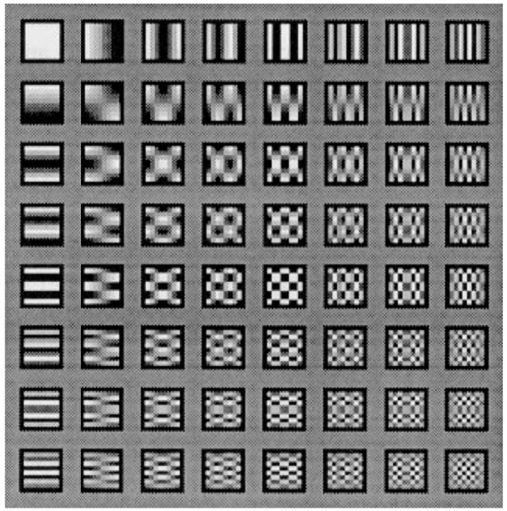
КЛАСИФІКАЦІЯ ТА СТИСНЕННЯ ЗОБРАЖЕНЬ

Мережа класифікації



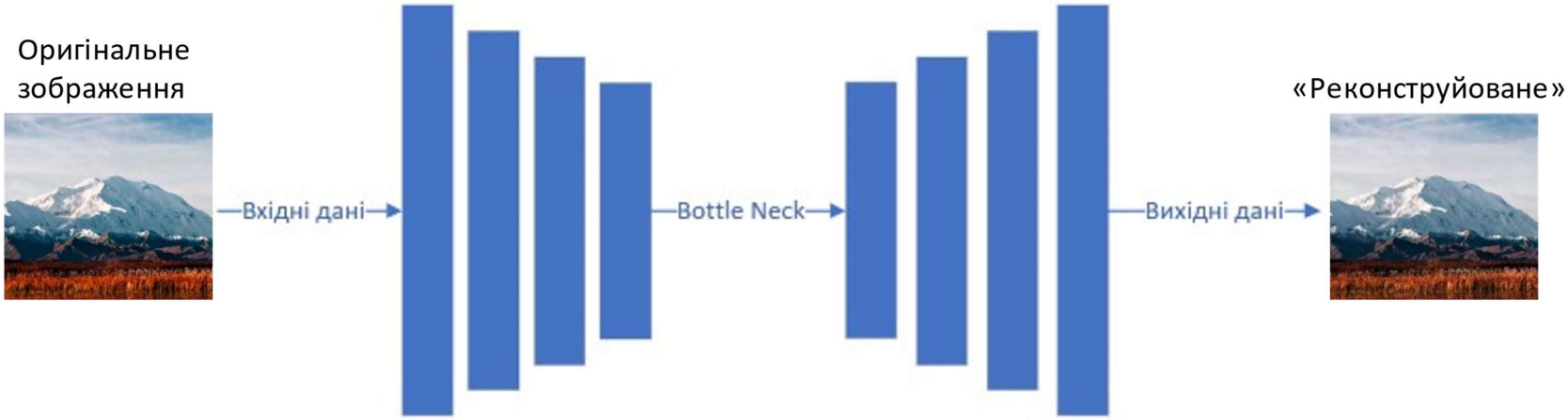
Згорткові фільтри

JPEG Стиснення

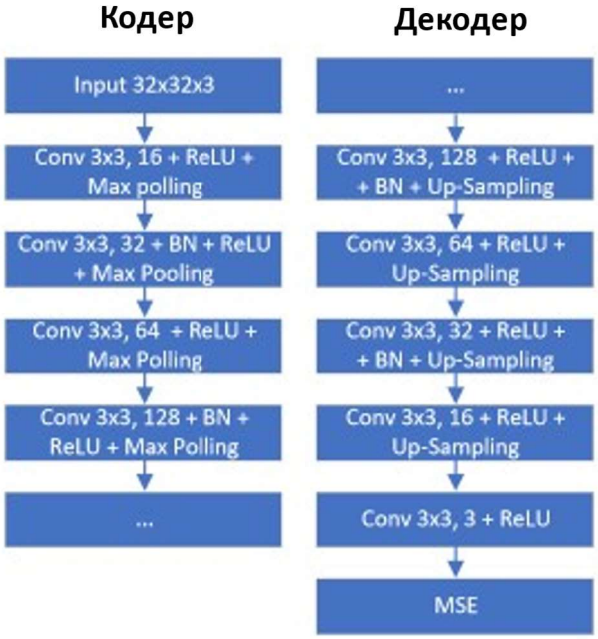


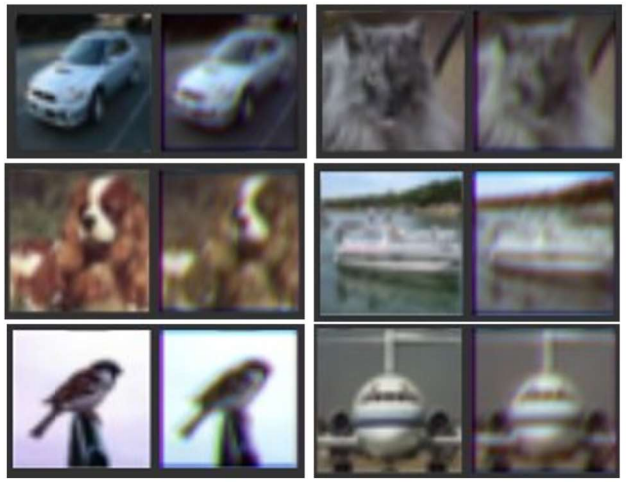
Фільтри DCT

BOTTLE NECK АРХІТЕКТУРА



МОДЕЛЬ МЕРЕЖІ СТИСНЕННЯ ЗОБРАЖЕНЬ





Оригінал - реконструкція