Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

за спеціальністю 123 «Комп'ютерна інженерія»

Тема наукової роботи:

КОМП'ЮТЕРНА СИСТЕМА ДЛЯ ФІЛЬТРАЦІЇ ТА РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ НА ОСНОВІ НЕЙРОМЕРЕЖЕВИХ ТЕХНОЛОГІЙ

Виконав _____ М. Р. Кутовий

Керівник роботи _____ А. І. Купін

Нормоконтроль _____ Д. І. Кузнєцов

Завідувач кафедри _____ А. І. Купін

Кривий Ріг
2024

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Ступінь вищої освіти магістр
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри, голова циклової комісії

_____ А. І. Купін

"____" _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

_____
(прізвище, ім'я, по батькові)

1. Тема роботи _____
_____
_____

керівник роботи _____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "___"_____ 20__року №__

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи _____
_____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_____
_____
_____
_____
_____
_____

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових
_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 101 сторінки , 77 рисунків, 9 таблиць, 2 додатки , 34 використаних джерел.

Мета – покращення ефективності розпізнавання зображень на основі використання згорткових нейронних мереж .

Проект складається з 4 розділів.

У першому розділі описуються основні передумови досліджень штучних нейронних мереж. Наведено огляд базових складових архітектури згорткової нейронної мережі, аспектів її навчання та розповсюджені проблеми . Розглянуті найкращі архітектури згорткових мереж для задач класифікації зображень

Другий розділ присвячено автокодувальникам та принципам їх роботи. Проведено регресіний аналіз для визначення факторів, що найбільше впливають на якість відновлення зображення автокодувальником. Розглядаються модифікації активаційної функції та агрегувального шару .

У третьому розділі проводиться вибір мови програмування, середовища розробки, та проектування архітектур нейронних мереж .Проводиться їх навчання та налаштування.

Четвертий розділ присвячено тестування програми. Для полегшення роботи з програмою створюються інструкції користувача і програміста, в яких описуються системні вимоги, користувацький інтерфейс та інше.

Explanatory note: 101 pages, 77 figures, 9 tables, 2 appendix, 34 sources used.

The goal is to improve image recognition performance through the use of convolutional neural networks.

The project consists of 4 sections.

The first section describes the basic prerequisites for the study of artificial neural networks. An overview of the basic components of a convolutional neural network architecture, aspects of its learning, and common problems are presented. The best convolution network architectures for image classification problems are considered

The second section is about auto-encoders and how they work. A regression analysis was performed to determine the factors that most influence the quality of image encoder recovery. Modifications of the activation function and the aggregation layer are considered.

The third section discusses the programming language, development environment, and design of neural network architectures. They are trained and adjusted.

The fourth section is devoted to testing the program. To help you work with the program, user and programmer instructions are created that describe the system requirements, user interface, and more.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 8
ВСТУП 9
1 ТЕОРЕТИЧНІ ОСНОВИ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ 11
1.1 Передумови розробки 11
1.2 Задача обробки зображень 11
1.3 Штучні нейронні мережі 14
1.4 Згорткові нейронні мережі 18
1.5 Розповсюджені проблеми згорткових нейронних мереж 28
1.6 Огляд існуючих нейромереж для розпізнавання зображень 30
Висновки за розділом 33
2 МОДИФІКАЦІЯ АРХІТЕКТУРИ НЕЙРОННИХ МЕРЕЖ 34
2.1 Згорткові автокодувальники 34
2.2 Модифікація нейронної мережі 41
2.2.2 Активаційна функція 46
Висновки за розділом 52
3 РОЗРОБКА ДОДАТКУ ДЛЯ ФІЛЬТРАЦІЇ ТА РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ 53
3.1 Концепт майбутньої програми 53
3.2 Вибір фреймворку для машинного навчання 56
3.3 Опис навчальної вибірки 59
3.4 Створення та навчання нейронних мереж 61
3.4.1 Мережа для розпізнавання 61
3.4.2 Мережа для реставрації 63
3.4.3 мережа для знешумлення 65
Висновки за розділом 68
..... 69

4.1 Інструкція користувача..... 69
4.2 Інструкція програміста 71
4.3 Тестування нейронних мереж..... 72
Висновки за розділом..... 76
ВИСНОВКИ 77
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 78
Додаток А..... 81
..... 92

ПЕРЕЛІК СКОРОЧЕНЬ

ШНМ – штучна нейронна мережа;
CAE convolutional autoencoder;
CNN c onvolutional neural network;
DAE denoising autoencoder;
ReLU rectified linear unit;
SGD Stochastic gradient descent.

ВСТУП

Актуальність роботи. У сучасному світі нейронні мережі мають колосальне охоплення. Вчені вважають дослідження, що проводяться в області вивчення поведінкових особливостей і станів нейронних мереж, вкрай перспективними. Перелік областей, в яких нейронних мереж знайшлося застосування, величезний. Це і класифікація образів, і прогнозування, і рішення апроксимаційних задач, і деякі аспекти стиснення даних, аналізу даних і, звичайно, застосування в системах безпеки різного характеру.

Дослідження нейронних мереж сьогодні активно відбувається в наукових спільнотах різних країн. При подібному розгляді вона представлена в якості окремого випадку ряду методів розпізнавання образів, дискримінантного аналізу, а також методів кластеризації.

Для вирішення цих задачі досить добре підходить використання згорткових нейронних мереж.

Мета дослідження. Метою даної роботи є поліпшення якості розпізнавання та фільтрації зображень різного рівня пошкоджень на основі використання згорткових нейронних мереж .Відповідно до поставленої мети визначено такі основні завдання дослідження:

1. Провести теоретичний огляд згорткових нейронних мереж для задач класифікації; особливості їх навчання та проблеми, що пов’язані з проектуванням, навчанням та вибором навчальних даних;
2. Провести огляд та дослідження згорткового автокодувальника для реставрації та зниження зашумленності зображень ;
3. Вдосконалити архітектури згорткових нейронних мереж на основі використання модифікованого агрегувального шару та модифікованої активакційної функції ;
4. Продемонструвати роботу нейронних мереж для фільтрації, реставрації та розпізнавання зображень шляхом створення програмного забезпечення .

Об’єкт дослідження – процес фільтрації та якісного розпізнавання зображень.

Предмет дослідження – методи, моделі та алгоритми для вирішення задачі якісної обробки зображень, покращення агрегувального шару та активаційної функції.

Наукова новизна. Доведено доцільність використання іншої активаційної функції для згорткового шару та доцільність використання змішанного агрегувального шару та змішаного агрегувального шару з додаванням невеликої долі випадковості на основі нормального розподілу для задач фільтрації зображень в межах однієї архітектури .Показано ефективність використання саме модифікованої архітектури .

Практичне значення отриманих результатів роботи полягає в тому, що конкретні модифікації нейронних мереж можна застосувати для підвищення показників роботив інших мережах .

Апробація результатів роботи та публікації. Матеріали досліджень, представлені в даній роботі, було представлено та опубліковано у збірнику тез доповідей на науково-технічній конференції: Всеукраїнська науково-технічна конференція студентів, аспірантів та молодих вчених «Комп'ютерні інтелектуальні системи та мережі» (КІСМ, м. Кривий Ріг, 2023-2024).

Структура та обсяг роботи. Робота складається зі вступу, 4 розділів, висновків, додатків, списку використаних джерел. Повний обсяг роботи складає 101 сторінку, 77 рисунків, 9 таблиць , 2 додатків та список використаних джерел зі 34найменуваннями на трьох сторінках.

Отже, було обґрунтовано актуальність роботи, сформульовано мету та задачі дослідження, наукову новизну та практичне значення

1ТЕОРЕТИЧНІ ОСНОВИ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

1.1 Передумови розробки

Важливість розпізнавання зображень у сучасному світі неможливо переоцінити. Facebook вже давно розпізнає всіх ваших друзів і автоматично відзначає їх, як тільки ви завантажуєте їх фото. Це один із прикладів того, як технології навколо стають тільки кращими і скоро переможуть людей у цій сфері. Система розпізнавання обличь – одна з тих дивовижних технологій, де машина може діяти майже так само розумно, як і людина, тобто вона може розпізнавати її обличчя і відрізняти від інших людей.

В той час, як алгоритми глибокого навчання можуть працювати нарівні аналогічному до звичайного людского мислення, хоча якість роботи у таких мережах поступово погіршується. Виявлено, що наявність надійних артефактів суттєво вплинути на точність розпізнавання сучасних підходів. Наявність реальних додатків, таких як пошуково-рятувальний дрон або автономна система керування, не спрацьовує при наявності навколишніх збурень, таких як дощ, туман або навіть розмитість, що викликається рухом і може мати несприятливі наслідки. І все це є реальними проблемами.

З огляду на таку ситуацію, було вирішено розробити програмний додаток, для розпізнавання та фільтрації зображень на основні штучних нейронних мереж, а саме ШНМ згорткової архітектури. Слід відмітити, що таке вирішення проблеми теж має свої недоліки, адже окрім необхідності мати певне технічне обладнання. Хоча і ця проблема в деякій мірі вирішується використанням заздалегідь навчених мереж.

1.2 Задача обробки зображень

В загальному значенні, обробка зображення - це маніпулювання зображенням, щоб покращити його або витягнути з нього інформацію. Існує два типи способів обробки зображень:

- Аналогова обробка зображень, яка використовується для обробки фотографій, роздруківки та інших копій зображень;
- Цифрова обробка зображень, яка використовується для маніпулювання

В обох випадках вхід – це зображення. Для аналогової обробки зображення вихідним є завжди зображення. Однак для цифрової обробки зображень на виході може бути зображення або деякі особливості та характеристики, пов'язані з цим зображенням.

Обробка зображення включає декілька ключових моментів:

- Збір зображення – це процес захоплення зображення датчиком і перетворення його в керовану сутність.
- Покращення зображення – покращує якість вхідного зображення та витягує з нього приховані деталі.
- Відновлення зображення видаляє всі можливі пошкодження (розмиття, шум або неправильне фокусування камери) із зображення, щоб отримати більш чисту версію. Цей процес заснований здебільшого на імовірнісних та математичних моделях.
- Обробка кольорових зображень включає обробку кольорових зображень та різних кольорових просторів. Залежно від типу зображення, ми можемо говорити про псевдокольорову обробку (коли кольорам присвоюються значення сірого масштабу) або обробку RGB (для зображень, які отримуються за допомогою кольорових датчиків або камер).
- Стиснення та декомпресія зображення дозволяють змінювати розмір та роздільну здатність зображення. Стиснення відповідає за зменшення цих розмірів і роздільної здатності, тоді як декомпресія використовується для відновлення зображень до оригіналу.
- Морфологічна обробка описує форму та структуру предметів на зображенні.
- Розпізнавання зображення - це процес ідентифікації конкретних зображень часто використовуються такі методи, як виявлення об'єктів, розпізнавання об'єктів та сегментація.

Під обробкою зображень в даній роботі роботі будуть розглядатися виключно відновлення зображення та його розпізнавання.

Більшість зображень, які отримуються тим чи іншим способом (наприклад, зроблені звичайними датчиками) потребують попередньої обробки, оскільки можуть містити занадто багато шуму в результаті чого бути спотвореними. Фільтрація – це найпоширеніший метод, який можна використовувати як для попередньої обробки, так і для обробки цифрових зображень [1].

Попередня обробка зображення – процес поліпшення якості зображення, що ставить за мету отримання на основі оригіналу максимально точного і адаптованого для автоматичного аналізу зображення.

Фільтрація використовується для поліпшення та зміни вхідного зображення. За допомогою різних фільтрів можна підкреслити або видалити певні функції зображення, зменшити шум зображення тощо.

Серед дефектів цифрового зображення можна виділити наступні види:

1. Цифрові аномалії.

2. Кольорові дефекти (недостатні або надлишкові яскравість і контраст, неправильний тон кольорів);
3. Розмитість.

Правила, які визначають фільтрацію (їх називають фільтрами), можуть бути найрізноманітнішими. Також фільтри, метою яких є прибирання шуму з зображення, орієнтовані на певну групу шумів[2]. Їх класифікація:

1. Шум «сіль і перець» - випадкові білі і чорні пікселі;
2. Імпульсний шум - випадкові білі пікселі;
3. Гаусів шум, що розподілений за нормальним законом.

Спотворення зображення, яке може виникнути при спробі зробити знімок тривимірного об'єкту реального світу, може мати досить складний характер, що залежить від умов навколишнього середовища (освітлення поверхні, атмосферні явища, тіні, сніг, дощ.). Математичні моделі, що намагаються наблизитись до реальності у питанні формуванні зображень, можуть містити у собі десятки та сотні рівнянь. Важливо розуміти, що у кожного пікселя зображення, значення яскравості не є незалежним і залежить формування яскравості навколишніх пікселів, тобто є залежність від законів геометрії. Спроба відновити таку складну модель з безліччю параметрів на основі використання математичного апарату є вже не тільки звичайною фільтрацією від шуму, а навіть реставрацією та реконструкцією[2].

Фільтрація зображень є однією з найбільш фундаментальних операцій комп'ютерного зору, розпізнавання образів і обробки зображень. Фактично, з тієї чи іншої фільтрації вихідних зображень починається робота переважної більшості методів. Зазвичай більшість матричних фільтрів можна класифікувати наступним чином: лінійні та нелінійні фільтри. Лінійні в свою чергу можна поділити на згладжувальні фільтри та фільтри для підвищення контрасту.

Щодо оцінок роботи самих фільтрів, то необхідно оцінювати спостережувану якість фільтрації за двома наступними критеріями:

- здатність фільтра видаляти (фільтрувати) з зображення шум;
- здатність фільтра зберігати на зображенні дрібнорозмірні деталі і форму контурів.

В даній роботі під попередньою обробкою буде матися на увазі як і відновлення зображення (фільтрація шуму), так і реставрація пошкоджених фрагментів зображення.

Реконструкція сигналу від пошкоджених або неповних вимірювань є важливим напрямом статистичного аналізу даних. Досягнення в глибинних нейронних мережах викликали значний інтерес уникнути традиційного, явного апріорного статистичного моделювання пошкоджень сигналу, а натомість навчитися відображати пошкоджені спостереження за чистими версіями.

Нейронні мережі почали використовуватися для обробки зображень порівняно недавно, тому на даний моментметоди усунення шуму з використанням нейромережевих алгоритмів нечисленні. З відомих наразі видів нейронних мереж, як

генеративно-змагальні мережі, які широко використовуються для синтезу складних зображень без участі людини.

Для вирішення задач класифікації (розпізнавання) та фільтрації (знешумлення та реставрація пошкодженних фрагментів) зображень було вирішено використовувати ШНМ згорткового типу. Для цього необхідно розглянути особливості роботи як звичайних ШНМ так і згорткових.

1.3 Штучні нейронні мережі

Область нейронних мереж спочатку була натхненна ціллю моделювання біологічних нейронних систем, але з тих пір розходилася і стала справою техніки та досягнення хороших результатів у завданнях машинного навчання. Для початку важливо розуміти основні та базові поняття, що стосуються нейронних мереж.

Основною обчислювальною одиницею як мозку так нейронних мереж є нейрон. У мозгу людини приблизно 87 млрд. нейронів, які з'єднані між собою приблизно 10^{14}-10^{15} синапсами[3,4]. На рисунку 1.1 наведено загальну математичну модель штучного нейрона з приведенням аналогії для біологічного нейрона.

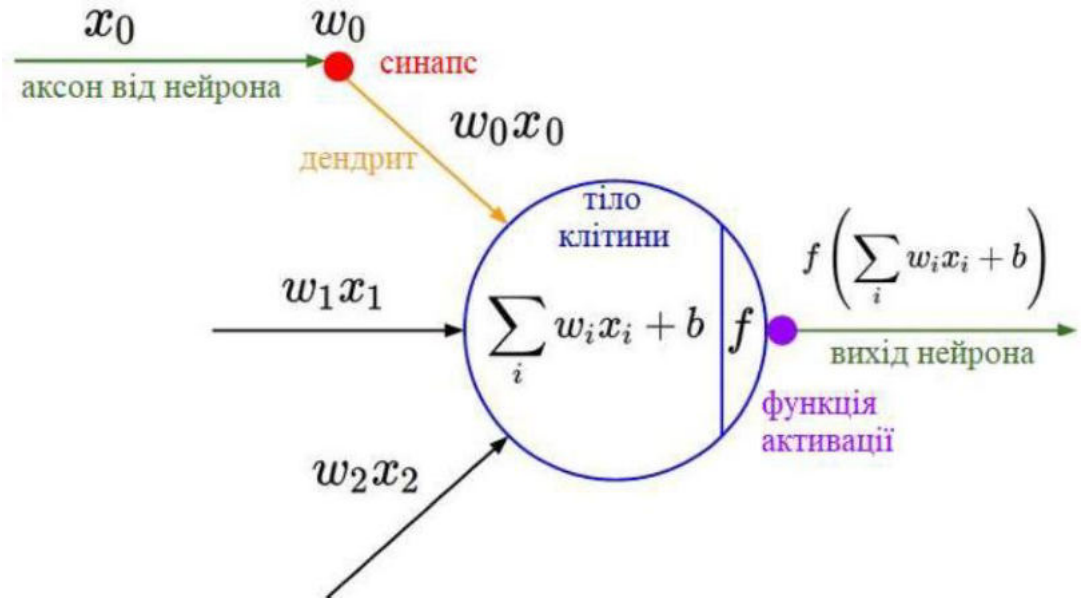


Рисунок 1.1 – Загальна математична модель штучного нейрона

У обчислювальній моделі нейрона, сигнали, що рухаються по аксонам (x_0), взаємодіють мультиплікативно (w_0*x_0) з дендритами інших нейронів на основі синаптичної сили в цьому синапсі (w_0). Ідея в тому, що синаптичні ваги w навчаються та контролюють силу впливу, тобто її напрямок, збудження або гальмування) одного нейрона на інший. У базовій моделі дендрити передають сигнал до тіла клітини, де всі вони підсумовуються. Потім застосовується активаційна функція, яка обчислює вихідний сигнал нейрона. Так склалося, що найбільш популярною функцією активації у нейронних мережах є

у межах від 0 до 1. Якщо зробити висновок, то кожен нейрон виконує добуток між входом та вагою, додає зміщення b і нелінійність (у даному випадку функцію активації).

Існує багато функцій активації, які додають нелінійність на виходах нейронів. Кожна з цих функцій приймає одне число на вході і виконує з ним певну математичну операцію. Існує декілька найбільш популярних активаційниї функцій. Це:

1. Логістична (сигмоїдальна);
2. Гіперболічний тангенс;
3. Випрямляч (ReLU);

Сигмоїда у формулі 1.1 є часто використовуваною активаційною функцією в классичних нейронних мережах[5]:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{1.1}$$

Вона приймає число на вході та і стискає його в діапазоні [0;1]. Наприклад, рисунок 1.2, великі від'ємні числа будуть дорівнювати 0, великі додатні числа – 1.

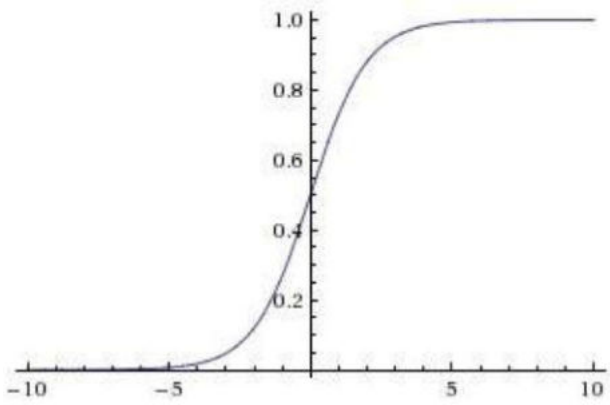


Рисунок 1.2 – Графік функції сигмоїди

Наступною активаційною функцією є гіперболічний тангенс (tanh), наведено у формулі 1.2. Якщо порівняти сигмоїду та гіперболічний тангенс, то можна сказати, що tanh є більш популярною функцією, оскільки вона є не тільки модифікованою сигмоїдою, але й перетворює від'ємні вхідні дані у від'ємні, а нулеві – у нульові[6].

$$\tan x = \frac{1 - e^{-2x}}{1 + e^{-2x}} = \frac{2}{1 + e^{-2x}} - 1, \tag{1.2}$$

Це відбувається тому, що центр виходу гіперболічного тангенсу знаходиться у точці 0. Тому на практиці активаційна функція гіперболічного тангенсу

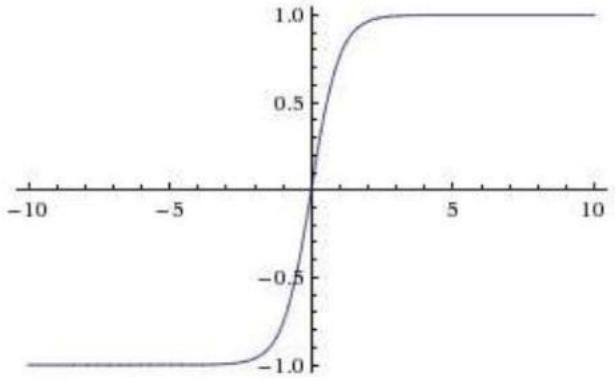


Рисунок 1.3 – Графік функції гіперболічного тангенсу

Випрямляч, або ReLU (Rectified Linear Unit) є найбільш популярною функцію саме для згорткових нейронних мереж (формула 1.3):

$$f(x) = \max(0, x), \tag{1.3}$$

На відміну від двох попередніх функцій, які приводили вхідне значення нулю. Дана активаційна функція не змінює вхідні дані, що більше або дорівнюють нулю, але все одно вона вносить нелінійність. Також плюсом є те, що операція знаходження максимуму двох елементів є більш ресурсо дешевою ніж експоненційні операції[7].

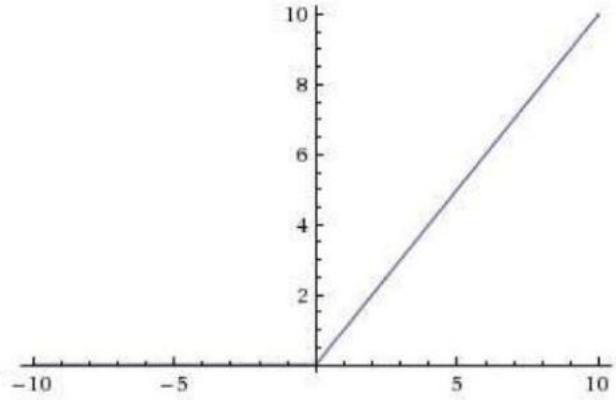


Рисунок 1.4 – Графік функції ReLU

Але варти зауважити, що головний недолік ReLU у тому, що всі негативні значення відразу стають рівними нулю, що знижує здатність моделі відповідати або навчатися на основі даних належним чином. Це означає, що будь-який негативний вхідний сигнал, наданий на вхід ReLU, негайно перетворює значення в нуль на виході, що в свою чергу, впливає на результат і неналежним чином враховує негативні значення, що отримуються під час [8].

Якщо спробувати узагальнити, то нейронна мережа є поєднанням з'єднаних між собою нейронів, зазвичай заснованих на використанні не дуже складних операцій. Нейрони в мережі надсилають сигнали іншим нейронам і ті в свою чергу також надсилають сигнали наступним нейронам. В решті решт, нейрони, що поєднані в мережу, в змозі вирішувати задачі, які складно або дуже важко формалізувати та вирішити на основі використання звичайних алгоритмів.

Організація штучних нейронів у нейронній мережі відбувається шарами. На рисунку 1.5 наведено організацію штучних нейронів за шарами. Зазвичай, нейрони першого шару є вхідними нейронами. Інформація, яку вони отримують є вхідною інформацією для мережі (зображення, аудіоданні, вектор). Вони в змозі (за необхідністю) обробити вхідну інформацію і надати її наступному шару. Другий шар нейронної мережі називається прихованим шаром, тому що він безпосередньо не зв'язаний ні з вхідною інформацією, ні з вихідною. Прихований шар виконує обробку інформації, яку отримав від попереднього шару і передає її нейронам вихідного шару. В даному випадку це є архітектурою найпростішої мережі, що складається з трьох шарів (вхідний, прихований та вихідний) і в результаті навчання цю мережу можна навчити знаходити взаємозв'язки у вхідній інформації.

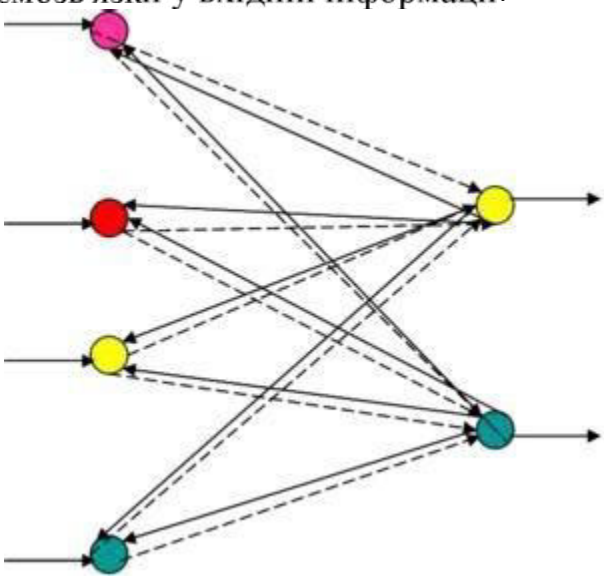


Рисунок 1.5 – Поєднання нейронів у шари нейронів

Щодо популярного у наші часи глибинного навчання, то цей термін можна використовувати лише по відношенню до більш великих і складніших нейронних мереж, що містять в собі більше ніж один прихований шар. При цьому шари, що виконують одні операції можуть чергуватися з іншими шарами, які виконують інші операції. В результаті чого зростає ступінь узагальнення і наступний шар намагається шукати взаємозв'язки у попередніх. В результаті чого ШНМ окрім знаходження простих взаємозв'язків здатна знаходити не тільки складні взаємозв'язками, а й взаємозв'язки між нейронами тощо [9].

1.4 Згорткові нейронні мережі

У штучних нейронних мережах, згорткова нейронна мережа (Convolutional Neural Network або CNN) є однією з основних архітектур для розпізнавання та класифікації зображень. Виявлення об'єктів, розпізнавання обличь, тощо - одні з напрямків, де широко використовуються CNN [10,11].

Наприклад, CNN, що навчена для класифікації зображень і приймає вхідне зображення на вході, обробляє його та класифікує за певними категоріями (собака, кішка, птах, т. п.). Нейронні мережі бачать вхідне зображення як масив пікселів, і це залежить від роздільної здатності зображення. Приклад зображення, що має 3 складових канали наведено на рисунку 1.6. Виходячи з роздільної здатності зображення, він побачить $h * w * d$ (h = висота, w = ширина, d = кількість вімирів).

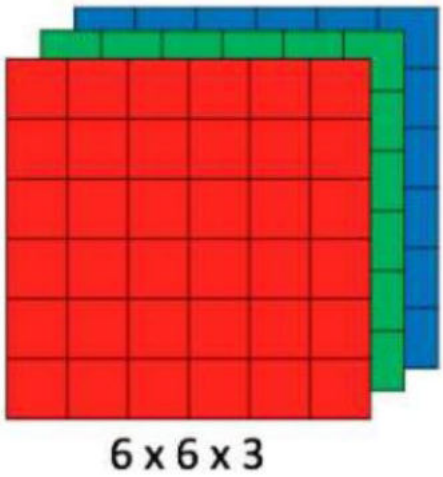


Рисунок 1.6 – Масив матриці RGB вхідного зображення

Технічно, CNN навчається та тестується, коли кожне вхідне зображення пройде скрізь його серію згорткових шарів з фільтрами (ядра), агрегувальних шарів, повноз'єднаних шарів (FC) та застосує функцію Softmax для класифікації об'єкта з імовірнісними значеннями між 0 та 1. На рисунку 1.7 представлений повний потік CNN для обробки вхідного зображення та класифікації об'єктів на основі значень.

Рисунок 1.7 – Структура CNN

Згортковий шар - це перший шар, який витягує функції із вхідного зображення. Згортка зберігає зв’язок між пікселями, вивчаючи особливості зображення, використовуючи невеликі квадрати вхідних даних.

Кожен згортковий шар містить серію фільтрів, відомих як згортковий фільтр. На рисунку 1.8 наведено приклад згорткових фільтрів. Фільтр - це матриця цілих чисел, які використовуються для підмножини вхідних значень пікселів, такого ж розміру, як ядро. Кожен піксель множиться на відповідне значення в ядрі (рисунок 1.9), потім результат підсумовується за одним значенням для спрощення обчислень[12].

У комп'ютерному зорі входом часто є 3-канальне RGB зображення. Для спрощення можна взяти зображення у відтінках сірого, що має один канал (двовимірну матрицю) та згорткове ядро 3x3. Ядро рухається по вхідній матриці чисел, і рухається по горизонталі стовпець за стовпцем, скануючі перші рядки в матриці, яка містить значення пікселів зображень. Потім ядро просувається вертикально до наступних рядків. Варто зазначити, що фільтр може проходити декілька пікселів одночасно.

Виконання згортки на зображеннях замість підключення кожного пікселя до кожного нейрона у нейронної мережі має дві основні переваги:

1. Зменшення кількості параметрів, які необхідно вивчити. Замість того, щоб вивчати ваги, що з'єднують кожен вхідний піксель, потрібно лише вивчити ваги фільтра (зазвичай це набагато менше, ніж вхідне зображення);

2. Зберігає місцезнаходження ознаки. Не потрібно стискати матрицю зображення у вектор, таким чином відносні положення пікс зберігаються. Наприклад, зображення кота. Інформація, що складає кота, включає відносні положення її очей, носа та пухнастих вух. Відбувається втрата розуміння, якщо представляти зображення як один довгий рядок чисел.

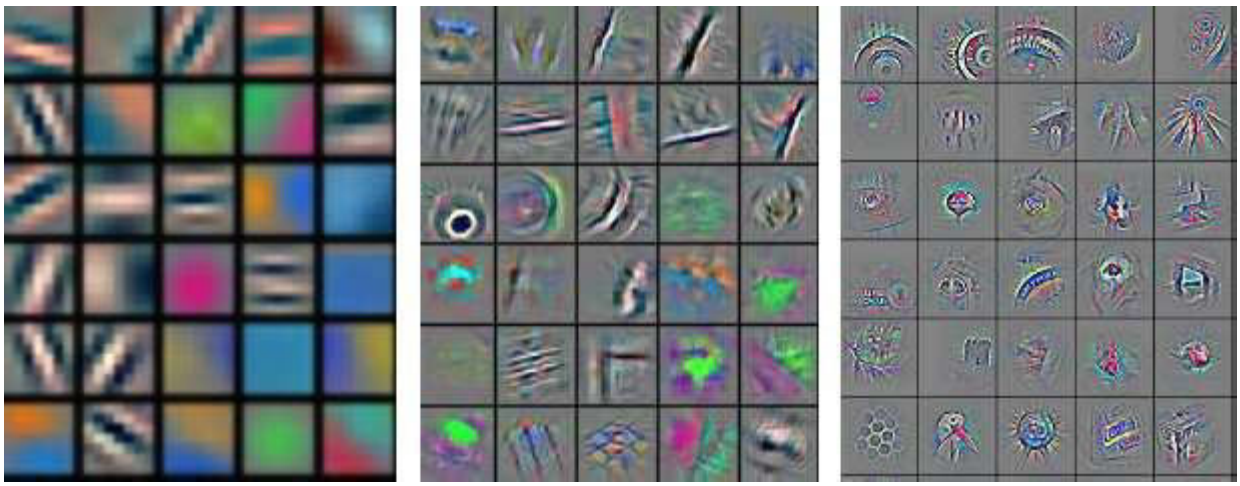


Рисунок 1.8 – Згортка на різних шарах нейронної мережі

Інтуїтивно можна уявити кожне згортання фільтра на зображенні як збір та обробку інформації на одній частині зображення та підсумовуваня їх

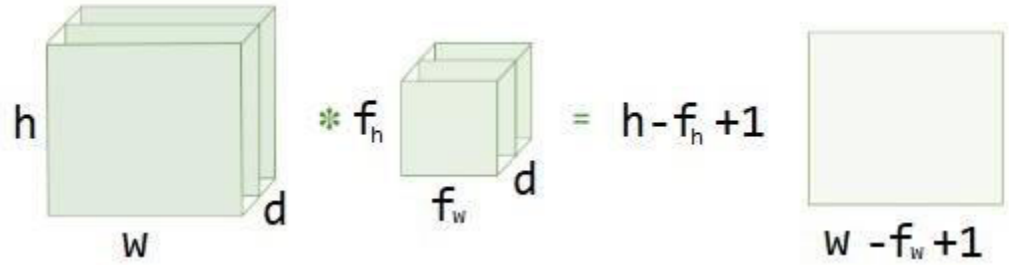


Рисунок 1.9 – Матриця зображення множиться на матрицю фільтра

На рисунку 1.9 (h * w * d) – вхідне зображення, (f_h * f_w * d) – фільтр, (h-f_h + 1) * (w * f_w + 1) * 1 – вихідний масив.

Також важливе значення при обробці вхідного зображення мають відступи та кроки. Кроки - кількість зрушень пікселів над вхідною матрицею. Коли крок дорівнює 1, відбувається переміщення фільтра на 1 піксель одночасно. Коли крок 2, тоді переміщення фільтра одночасно на 2 пікселі, тощо[13]. Приклад кроку наведено на рисунку 1.10.

Рисунок 1.10 – Крок у 2 пікселі

В свою чергу, відступи слугують іншій цілі. Іноді фільтр не підходить по розмірам ідеально для вхідного зображення. Є два варіанти, вирішення цієї проблеми: додати нулі там, щоб зображення підходило по розміру або відкинути частину зображення там, де фільтр не вміщається повністю. Це називається допустимим відступом, який при цьому зберігає лише дійсну частину зображення.

Згортання зображення з різними фільтрами може виконувати такі операції, як виявлення країв, розмиття та різкість, застосовуючи фільтри. Наведений нижче приклад показує різні зображення згортки після [14].

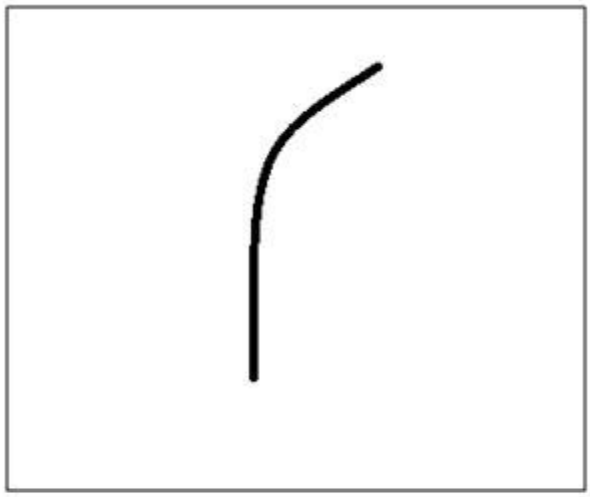
Операція	Фільтр	Результат
Тотожність	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Виявлення країв	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Різкість	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Розмивання (нормальне)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Розмивання Гауса	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Рисунок 1.11 – Найбільш поширені фільтри

Кожен з цих фільтрів може розглядатися як ідентифікатор ознак зображення. Коли мова йдеться про ознаки, маються на увазі такі речі, як прямі края, прості кольори та криві лінії. Якщо ще простіше, то це можна назваті як усі найпростіші характеристики, які є спільними майже для будь-якого зображення. Наприклад, перший фільтр 7 x 7 буде детектором кривої (див. рис. 1.12).

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Піксельне представлення фільтру



Візуалізація фільтру для пошуку кривої лінії

Рисунок 1.12 – Фільтр згорткового шару

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Рисунок 1.13 – Шаблон 1

В результаті отримаємо: (50*30)+(50*30)+(50*30)+(20*30)=6600. Це є доволі великим числом і це призведе до того, що нейрон з фільтром на виході активується. Тоді можна сказати, що якщо у вхідного зображенні є форма, яка представлена у конкретному фільтрі, то після множення та сумування разом на виході отримаємо високе значення.

Тепер повернемося до математичної візуалізації. Коли у нас є цей фільтр у верхньому лівому кутку вхідного обсягу, він обчислює множення між фільтром і значеннями пікселів в цій області. Тепер візьмемо приклад зображення, яке необхідно класифікувати.

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

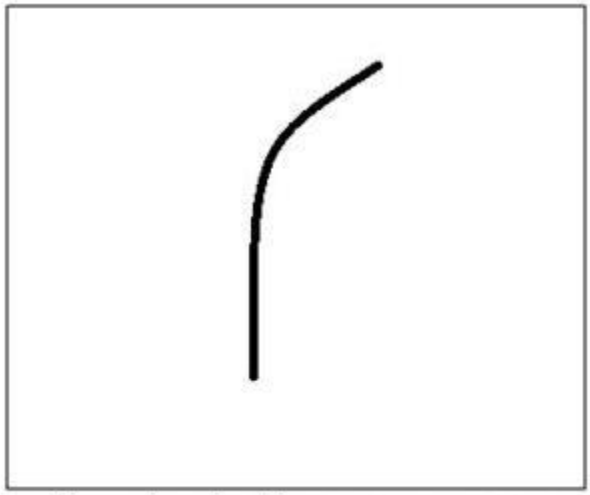
*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Рисунок 1.14 – Шаблон 2

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Піксельне представлення фільтру



Візуалізація фільтру для пошуку кривої лінії

Рисунок 1.12 – Фільтр згорткового шару

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Рисунок 1.13 – Множення фільтру на частину вхідної матриці зобрження

В результаті отримаємо: (50*30)+(50*30)+(50*30)+(20*30)=6600. Це є доволі великим числом і це призведе до того, що нейрон з фільтром на виході активується. Тоді можна сказати, що якщо у вхідного зображенні є форма, яка представлена у конкретному фільтрі, то після множення та сумування разом на виході отримаємо високе значення.

Тепер повернемося до математичної візуалізації. Коли у нас є цей фільтр у верхньому лівому кутку вхідного обсягу, він обчислює множення між фільтром і значеннями пікселів в цій області. Тепер візьмемо приклад зображення, яке необхідно класифікувати.

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Рисунок 1.14 – Множення фільтру на частину вхідної матриці зобрження

В результаті множення цих матриць буде 0. Значення набагато нижче. Це тому, що в секції зображення не було нічого, що відповідало б фільтру детектора кривої. Важливо розуміти, що виходом цього згорткового шару є карта активації нейронів. Таким чином, в простому випадку згортки з одним фільтром (і якщо цей фільтр є детектором кривих), карта активації покаже області, в яких, швидше за все, будуть криві на малюнку.

Після згорткового шару, йде агрегувальний шар. Він зменшує кількість

- зивається зибіркою або зменшенням розміщення, що зменшує розмірність кожної карти, але зберігає важливу інформацію [15]. Він може бути різних типів:

1. Максимальна агрегація;
2. Середня агрегація;
3. Сумарна агрегація.

Агрегувальний шар з функцією максимального значення обирає найбільший елемент з мапи ознак (рисунок 1.15). Середня – середнє арифметичне всіх елементів. Сумарна – сума елементів усіх елементів мапи ознак.

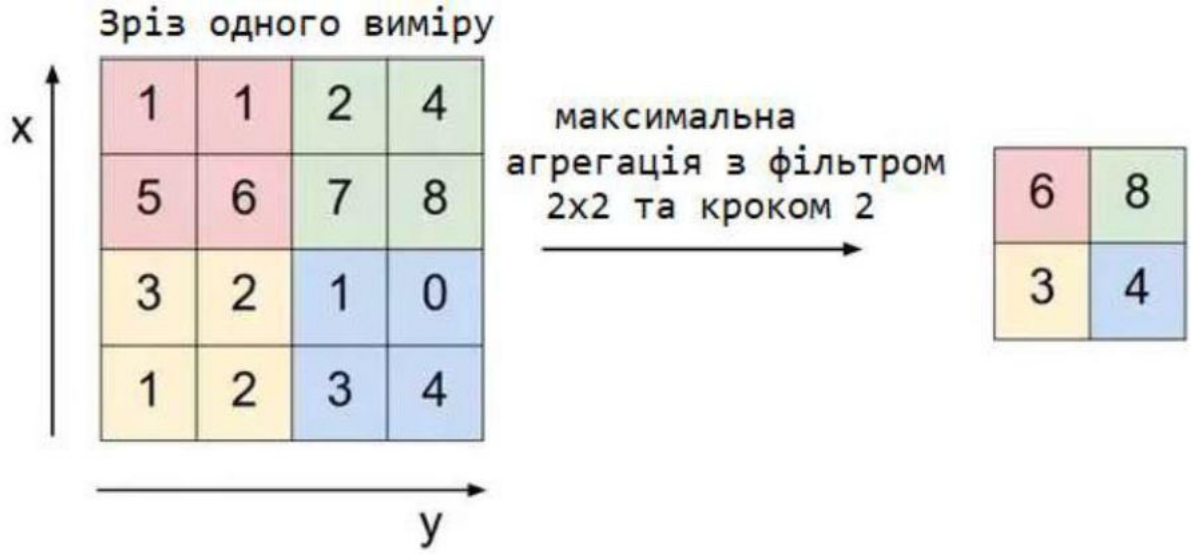


Рисунок 1.15 – Максимальна агрегація

Шар зрізаних лінійних вузлів або Rectified Linear Units (ReLU) є активаційною функцією, про яку вжу було зазначено у попереднбому підрозділі. Вона визначена наступним чином:

$$f(x) = x^{+} = \max(0, x), \qquad (1.5)$$

де x – вхідне значення нейрону.

Станом на 2019 рік, є найбільш популярною передавальною функцією для глибинних нейронних мереж. Результат роботи ReLU наведено на рисунку 1.16. Мета ReLU полягає у впровадженні нелінійності у згортковій нейронній мережі.

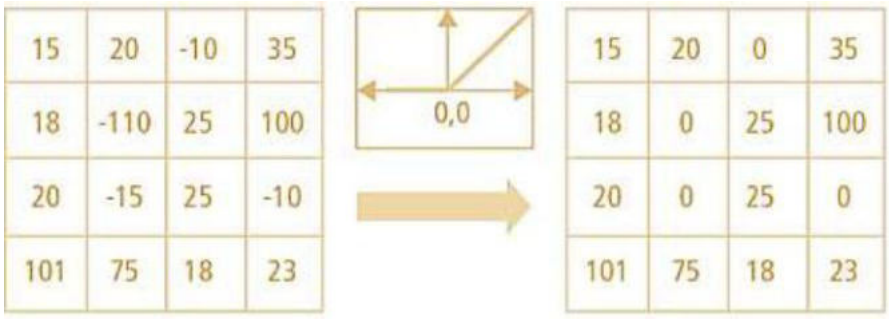


Рисунок 1.16 – Результат роботи ReLU

Далі йде шар, який називається повнозв'язним шаром (FC), рисунок 1.17. Перед цим відбувається перетворення матриці у вектор і подання на вхід повнозв'язного шару, який являє собою штучну нейронну мережу прямого поширення, або звичайний перцептрон.

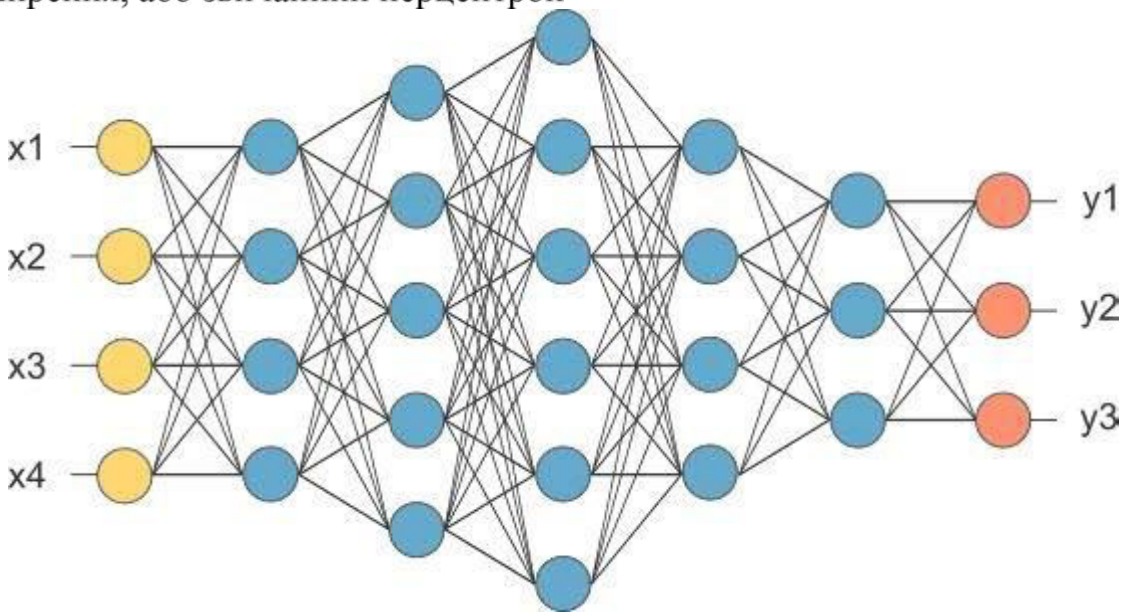


Рисунок 1.17 – Після агрегувального шару та стискання матриці у вектор, вектор подається на вхід повнозв'язної мережі

На наведеній зображенні, матриця карти ознак зображення була перетворена у вектор (x1, x2, x3,…). У повнозв'язному шарі відбувається поєднання цих ознак разом для створення моделі. В цій мережі зазвичай використовуються такі функції активації, як софтмакс чи сигмоїда, щоб якісно класифікувати результати і віднести до одного з класів (наприклад, кішка, собака, літак, машина).

Тобто, якщо підсумувати усе вищесказанне, для побудови згорткової нейронної мережі необхідно:

1. Отримати вхідне зображення на згортковому шарі;
2. Обрати параметри, застосувати фільтр з певним кроком та відступом (якщо потребується);
3. Виконати згортку на зображенні та застосувати ReLU для отримання нелінійности;
4. Верифікувати;

5. Додати стільки згорткових шарів, скількі необхідно для конкретної задачі;
6. Перетворити отриману матрицю у вектор та подати на повнозв'язну нейронну мережу;
7. Як результат, отримати клас, використовуючи активаційну функцію (наприклад, логістична регресія з функцією втрат) та класифікувати зображення.

Перед запуском CNN, значення ваг або фільтрів випадкові. Фільтри не знають як шукати краї та криві. Фільтри у вищих шарах не знають, як шукати більш складні фігури.

Процес навчання штучних нейронних мереж розглядається як налаштування архітектури (якщо необхідно) і ваг зв'язків між нейронами (параметрів) для ефективного виконання поставлених перед мережею завдань.

Для навчання нейронних використовується алгоритм зворотнього поширення помилки (Back Propagation). Цей метод навчання багатошарової нейронної мережі називається узагальненим дельта-правилом. Метод був запропонований в 1986 р Румельхартом, Макклеланд і Вільямсом. Це ознаменувало відродження інтересу до нейронних мереж, який став згасати на початку 70-х років. Даний алгоритм є першим і основним практично використовуємим для навчання багатошарових нейронних мереж.

Зворотне розповсюдження можна розділити на 4 окремі секції: передній прохід, функцію втрат, зворотний прохід та оновлення ваг. Під час переходу вперед знімається навчальний образ, який є масивом чисел і передається по всій мережі. Оскільки всі значення ваг або фільтрів були випадково ініціалізовані, в основному вихід не буде надавати перевагу жодному номеру. Мережа, з її поточною вагою, не в змозі шукати ті низькорівневі функції або, таким чином, не може зробити жодного розумного висновку щодо того, якою може бути класифікація. Наступний крок це функція втрат, яка є частиною зворотного розповсюдження. Наприклад, першим введеним навчальним зображенням було позначення 3. Мітка для цього зображення буде [0 0 0 1 0 0 0 0 0 0]. Функцію втрат можна визначити різними способами, але загальною є MSE (середня квадратична помилка):

$$E_p = \frac{1}{2}\sum_j (t_{pj} - y_{pj})^2 \ , \qquad (1.6)$$

де, E_p – величина функції помилки для вхідного образу p;
t_{pj} – бажаний віхід нейрона j для вхідного образу p;
y_{pj} – активований вихід нейрона j для вхідного образу p.

В загальному випадку функція втрат використовується для розрахунку помилки між реальними і отриманими відповідями. Головна задача - мінімізувати цю помилку. Таким чином, функція втрат ефективно наближає [15].

Функція втрат вимірює «наскільки добре» нейронна мережа надає результати у порівнянні з навчальною вибіркою. Вона також може залежати від таких змінних, як ваги і зміщення.

Функція втрат одновимірна і не є вектором, оскільки вона оцінює, наскільки добре нейронна мережа працює в цілому.

Найбіль відомі функції втрат:

1. Квадратична (середньоквадратичне відхилення);
2. Крос-ентропія;
3. Експоненціальна (AdaBoost);
4. Відстань Кульбака - Лейблера або приріст інформації.

Среднеквадратічное відхилення - найпростіша фукция втрат і найбільш часто використовувана.

Функція втрат в нейронної мережі повинна відповідати двом умовам:

1. Функція втрат повинна бути записана як середнє;
2. Функція втрат не повинна залежати від будь-яких активаційних значень нейронної мережі, крім значень, які видаються на виході.

Важливо зазначити про методи оптимізації, що існує багато методів оптимізації помилки. Різні методи підходять для різних випадків, і в деяких випадках можуть навіть бути скомбіновані.

Популярні оптимізатори:

1. SGD: стохастичний градієнтний спуск з підтримкою імпульсу;
2. MSprop: адаптивний метод оптимізації швидкості навчання, запропонований Джеффом Хінтон;
3. Adam: адаптивна оцінка моментів (Адам), яка також використовує адаптивні швидкості навчання.

В якості методу мінімізації помилки в класичному прикладі використовується метод градієнтного спуску. Суть цього методу зводиться до пошуку мінімуму (або максимуму) функції за рахунок руху вздовж вектора градієнта. Для пошуку мінімуму рух має здійснюватися в напрямку антіградіента[16,17].

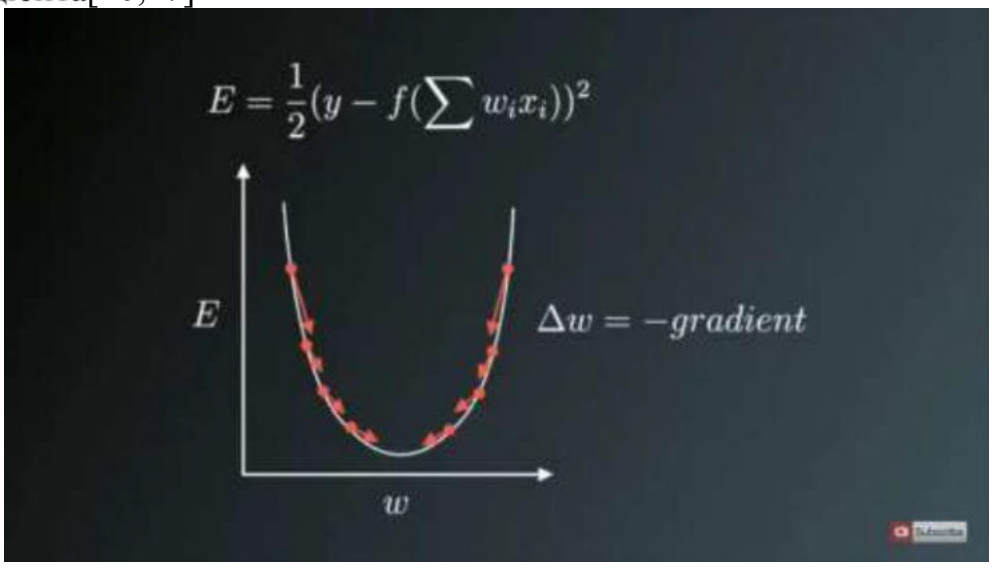


Рисунок 1.18 – Ваги оновлюються у протилежному напрямку

Градієнт функції втрати є вектором часткових похідних, обчислюється за формулою:

$$\nabla E(W) = \left[\frac{dE}{dw_1}, \dots, \frac{dE}{dw_n}\right], \quad (1.9)$$

де, $\nabla E(W)$ – градієнт функції втрат від матриці ваг;

$\frac{dE}{dw_1}$ – часткова похідна функції помилки за вагою нейрона;

n – загальна кількість ваг у мережі.

Похідну функції помилки конкретного вхідного образу можна записати наступним чином:

$$\frac{dE}{dw_{ij}} = \frac{\partial E}{\partial y_j} * \frac{\partial y}{\partial s_j} * \frac{\partial s_j}{\partial w_{ij}}, \quad (1.10)$$

де, $\frac{dE}{dw_{ij}}$ – значення похідної функції помилки за вагою w_{ij} між і та j нейронами;

$\frac{\partial E}{\partial y_j}$ – помилка нейрона j;

$\frac{\partial y}{\partial s_j}$ – значення похідної функції активації за її аргументом для нейрона;

$\frac{\partial s_j}{\partial w_{ij}}$ – вихід і нейрона попередньоо шару, по відношенню до нейрону j.

Для вихідного шару помилка визначена в явному вигляді.. Але для розрахунку помилки в прихованих шарах необхідно використовувати алгоритм зворотного поширення помилки. Його суть полягає полягає в послідовному обчисленні помилок прихованих шарів за допомогою значень помилки вихідного шару, тобто значення помилки поширюються по мережі в зворотному напрямку від виходу до входу.

Помилка δ для прихованого шару розраховується за формулою:

$$\delta_i = \frac{\partial y_i}{\partial s_i} * \sum_j \delta_j * w_{ij}\ , \quad (1.11)$$

де, δ_i – помилка нейрона і прихованого шару;

$\frac{\partial y_i}{\partial s_i}$ – значення похідної функції активації за її аргументом для нейр. j;

δ_j – помилка нейрона j наступного шару;

w_{ij} – вага зв'язку між нейроном і прихованого шару та нейроном j вихідного (або іншого прихованого) шару.

Алгоритм зворотнього поширення помилки зводиться до наступних етапів:

1. Пряме поширення сигналу по мережі, обчислення стану нейронів;
2. Обчислення значення помилки δ для вихідного шару;
3. Зворотне поширення: послідовно від кінця до початку для всіх прихованих шарів обчислюємо δ за формулою 1.11;
4. Обчислити кінцевий результат.

1.5 Розповсюджені проблеми згорткових нейронних мереж

В залежності від цілей, які переслідуються при побудові ШНМ, можна поставити декілька питань, які є важливими для більш детального розуміння можливості створення мережі:

1. Чи можна отримати достатню кількість якісних та специфічних даних для навчання моделі?
2. Чи відомо, яка архітектура ШНМ (її параметри, кількість та типів шарів, активаційна функція) найбільш усього відповідає поставленій задачі?
3. Чи потрібно обирати новітні нейронні мережі з глибинним навчанням та тисячами параметрів, що можна налаштувати, чи потрібно щось менш інтенсивне в плані обчислювальною потужності, що так само буде відповідати поставленим цілям?
4. Як уникнути проблеми перенавчання мережею?

Це лише деякі з питань, які повинні вирішити спеціалісти перед тим, як приступити до виконання завдань моделювання. Перш ніж згорткові моделі нейронних мереж (або будь-які алгоритми глибокого навчання) зможуть видавати прогнози, вони повинні бути ретельно навчені, в залежності від

вимагають, щоб великі обсяги помічених візуальних даних проходили через їх шари для досягнення високого рівня точності в задачах регресії і класифікації. Таким чином, їх етап навчання вимагає великих обчислювальних ресурсів. Крім того, виникають проблеми з отриманням чималих наборів даних за конкретними сферам.

Зазвичай використовуються графічні процесори для прискорення процесу створення параметрів з'єднань у згорткових нейронних мережах. Маючи кілька ядер і забезпечуючи розпаралелювання обчислень, ці процесори ідеально підходять для простих математичних математичних обчислень; в цьому відношенні вони набагато швидше звичайних центральних процесорів, які в машинному навчанні зазвичай використовуються для послідовних складних обчислень.

Під час навчання класифікатор моделі рухається вперед, проходячи через позначений набір даних (один повний прохід через всі об'єкти даних називається епохою) кілька разів. Вхідні зображення обробляються шарами мережі, які на даний момент мають довільні ваги.

Потім на основі отриманих значень модель обчислює помилку за допомогою функції втрат. Вона порівнює свої перші вихідні дані, результати випадкової ініціалізації, з фактичними мітками, щоб побачити, наскільки близько (або далеко) вони знаходяться від бажаних значень. Після цього помилка поширюється назад від кінця до початку мережі, щоб CNN могла відповідним чином оновити свої ваги. Це відбувається за допомогою алгоритму оптимізації градієнтного спуску (поновлення ваги).

Вага змінюється тільки один раз за епоху, і тому навчання моделі CNN

що набори даних для сучасних CNN мають тенденцію бути великими, потрібно багато часу і ресурсів, щоб пройти через кожну вибірку. Таким чином, щоб прискорити процес і скоротити витрати, зазвичай, дані розбивають на партії і змушують свої моделі оновлювати ваги після обробки кожної партії. Цей підхід вимагає набагато менше часу для зближення і, таким чином, зменшує загальну кількість епох, необхідних для навчанн я.

Наявність моделі, яка працює з партією вхідних вибірок (зображень, в разі CNN), одночасно допомагає запобігти переоснащення, а також дозволяє досягти більшої ефективності роботи за рахунок амортизації ваг завантаження з пам'яті графічного процесора за кількома входів.

Стадія виводу, з іншого боку, відноситься до випадку, коли CNN, досить навчений, застосовується до даних за межами позначеного набору даних; коли він робить прогнози на вихідні дані. Він також включає в себе розрахунок

яка використовується для виведення, більше не змінює свої параметри. Кількість об'єктів даних, переданих в CNN на цьому етапі, навмисно менше:

можливості графічних процесорів, коли мова йде про таких додатках, як конвеєри обробки зображень, які працюють з затримкою в реальному часі теж має значення. Може знадобитися багато часу, щоб об'єднати зображення в велику партію, в той час як завжди прагнуть скоротити загальний час відгуку своєї моделі. Отже, ключем тут є знаходження правильного балансу між пропускною спроможністю і затримкою; необхідно максимізувати придатні для використання партії і не допускати перевищення затримки певного порогового значення для конкретного додатка.

Як було вищезазначено, дуже складно зібрати достатньо даних для навчання конкретному алгоритму глибокого навчання. Особливо, коли ви працюєте над якоюсь вузькопрофільною проблемою. Одним із способів вирішення цієї проблеми є трансферне навчання. Не обов'язково навчати CNN з нуля; можна перепризначити модель, яка вже була навчена і має основний набір функцій. Це працює, тому що CNN, як правило, вибирають дуже загальні

потім використовуватися для інших випадків використання. Трансферне навчання допомагає зменшити обчислювальне навантаження і уникнути необхідності використовувати ряд графічних процесорів. Коли застостовується цей метод, не відбувається ініціалізація ваг випадковим чином на початку; використовуються параметри шару, які модель вже вивчила на інших даних. Потім можна приступати до подальшого навчання моделі в нашому наборі даних, який, як правило, не є вичерпним[18].

Є кілька способів зробити це:

- Продовжувати виконувати зворотне поширення і оновлювати ваги на попередньо навченій моделі або вибірково налаштовувати деякі її шари. Все починається з налаштування верхніх шарів. Потім поступово повертатися до більш ранніх рівнів, оцінюючи продуктивність мережі, щоб побачити, де необхідно працювати.

- Можна використовувати попередньо навчену мережу в якості екстрактора ознак і навчати машину опорних векторів або інші лінійні класифікатори за допомогою цих функцій. Цей метод хороший, коли не так багато доступних даних, і потрібно уникнути перенавчання.

Дуже заманливо використовувати глибинні та глибинні згорткові нейронні мережі для кожного завдання. Але це може бути поганою ідеєю, тому що:

- Обидві вимагають значно більшої кількості даних для навчання, щоб досягти мінімальної бажаної точності;
- Обидві мають експонентну складність;
- Занадто глибока нейронна мережа спробує зламати фундаментальні уявлення, але при цьому вона буде робити хибні припущення і намагатися знайти псевдо-залежності, які не існують;
- Занадто глибинна нейронна мережа буде намагатися знайти більше ознак, ніж є. Таким чином, подібно до попередньої, вона почне робити неправильні припущення про дані.

1.6 Огляд існуючих нейромереж для розпізнавання зображень

Побудова штучних нейронних мереж на якісному рівні вважається складною задачею, оскільки потрібно врахувати багато параметрів та не усі важливі параметри можливо налаштувати таким чином, щоб це давало задовільний результат. Тому важливо розглянути найвідоміші архітектури згорткових нейронних мереж, які досягли успіхів у задачі класифікаціх зображень. На рисунку 1.19 наведено графік, який ілюструє збільшення кількості шарів у моделях з часом, які показували найкращі у свій час результати в задачі класифікації зображень. Як видно з графіка, спостерігається очевидна тенденція до ускладнення архітектури моделей. Така еволюція зумовлена в першу чергу здешевленням та збільшенням потужності [19].

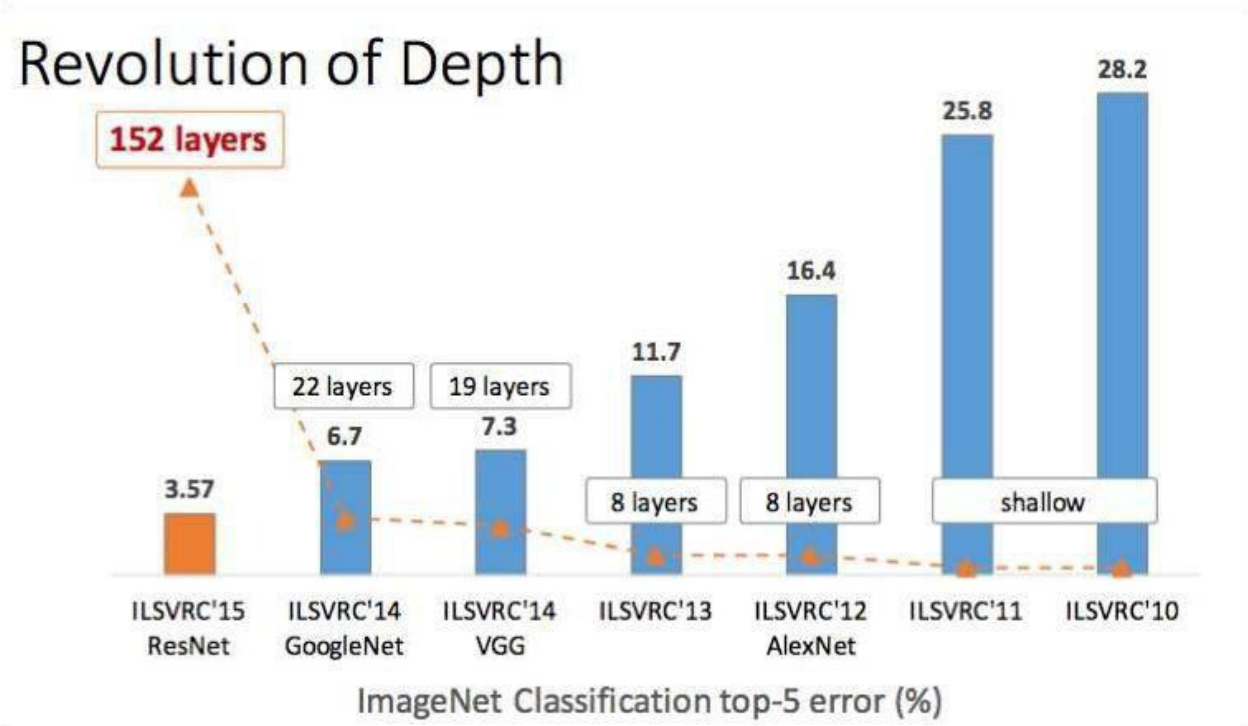


Рисунок 1.19 – Після агрегувального шару та стискання матриці у вектор, подається на повнозв'язний шар

Наведемо коротку характеристику найбільш поширених варіантів таких архітектур. Архітектуру AlexNet наведено на рисунку 1.20.

AlexNet був набагато більшим, ніж попередні CNN, які використовувались для виконання завдань із комп'ютерного зору. У нього 60 мільйонів параметрів і 650 000 нейронів, а навчалась мережа на двох графічних процесорах GTX 580 3 Гб. Загальний час навчання шість днів. Сьогодні існує набагато складніші CNN, які можуть працювати на швидших графічних процесорах дуже ефективно навіть на дуже великих наборах даних. Але для 2012 року це був прорив і доволі великою мереже [20].

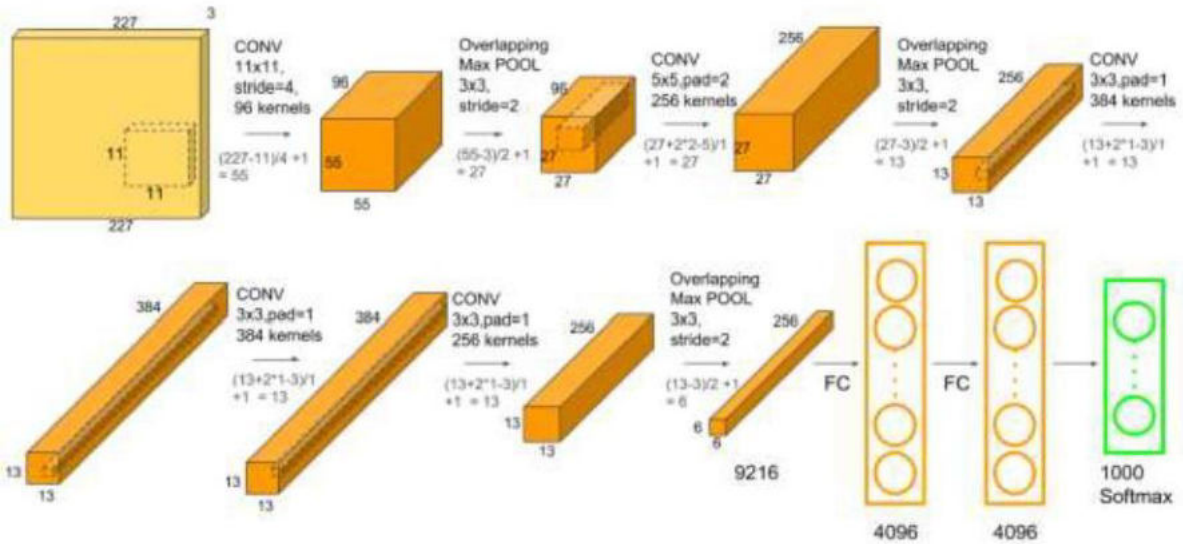


Рисунок 1.20 – [illegible] AlexNet

AlexNet складається з п’яти згорткових шарів і трьої повністю з'єднаних шарів. Кілька згорткових ядер отримують цікаві функції в зображенні. В одному згортковому шарі зазвичай є багато ядер однакового розміру. Наприклад, перший шар конвертів AlexNet містить 96 ядер розміром 11x11x3. Але зазвичай, ширина і висота ядра є однаковою, а глибина така ж, як і кількість каналів.

Нелінійність ReLU застосовується після всіх нейронів у згортковому шарі і повністю пов'язаних шарів. Нелінійність ReLU першого та другого шарів згортання супроводжується етапом локальної нормалізації перед об'єднанням. Але згодом дослідники не вважали нормалізацію дуже корисною. Тож ми не будемо детально розбиратися над цим.

Мережа VGG-16 (2014 р.), складається з 16 шарів, що мають ваги, які навчаються та 5 агрегувальних шарів. Є однією з найбільш популярних у науковій спільноті та знаходить велику кількість застосувань. Архітектуру VGG-16 наведено на рисунку 1.21. Головною перевагою є зменшення розмірів рецептивного поля до 3x3 пікселів та збільшення кількості шарів. Хоча з іншого боку, дана нейронна мережа має понад 140 мільйонів параметрів, що передабачає великі обчислювальні потужності для її навчання. Пізніше було створено модифікацію з 19 шарами (VGG-19). В ній додаються нові три згорткові шари [21].

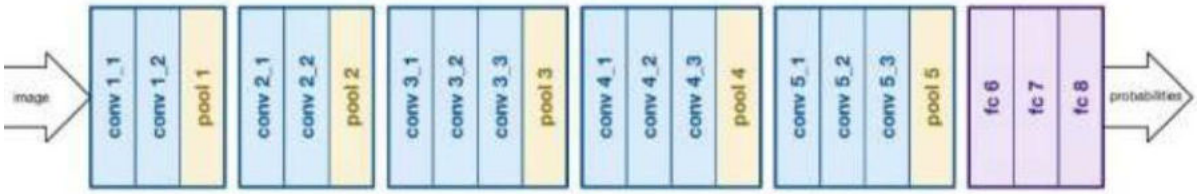


Рисунок 1.21 – Архітектура VGG-16

GoogLeNet перемогла у ILSVRC 2014 р., досягнувши помилки top-5 зі значеням у 6,67%, що можна вважати подібним до здібності розпізнавати людиною. Архітектуру GoogLeNet наведено на рисунку 1.22. В середені мережі було використано новітній шар inception. Мережа складаєься з 22 шарів. Кількість параметрів становить близько чьотирьох мільонів[22].

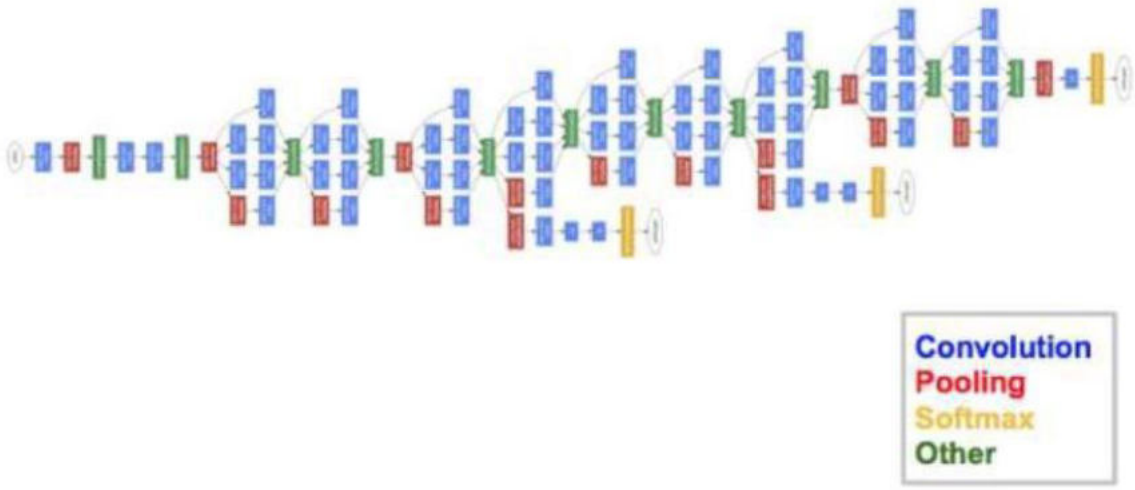


Рисунок 1.22– GoogLeNet

У ResNet вводиться нова архітектура з пропусками зв'язків (skip connections). Архітектуру ResNet наведено на рисунку 1.23. Окрім цього в мережі застосовується пакетна нормалізація. Завдяки використанні пакетної нормалізації та пропусків зв'язків мережа отримала можливість навчати 152 прихованих шари при значно меншій обчислювальній потужності, ніж у VGG. Мережа отримала у змаганні ILSVRC 2015 р. 3,57%, що перевершує людські показники [23].

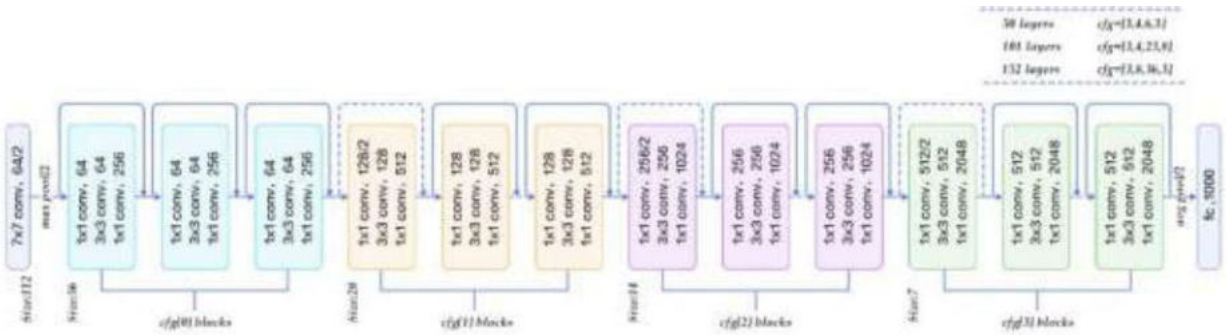


Рисунок 1.23 – Архітектура ResNet

Висновки за розділом

Отже, в даному розділі було наведено основні передумови створення прикладного додатку для обробки зображень на основі нейронних мереж. Проведено загальний огляд ШНМ, згорткових ШНМ та їх архітектури. Розглянуто проблеми, які зустрічаються у згорткових мережах та приклади існуючих архітектур, які займали призові місця. Розглянуто їх переваги, недоліки. У зв'язку з особливостями роботи майбутньої програми значна частина увага була приділена .

2 МОДИФІКАЦІЯ АРХІТЕКТУРИ НЕЙРОННИХ МЕРЕЖ

2.1 Згорткові автокодувальники

Автокодувальники (AutoEncoders) — це нейронні мережі прямого поширення, які відновлюють вхідний сигнал на виході. Всередині у них є прихований шар, який представляє собою код, що описує модель. Автокодувальники конструюються таким чином, щоб не мати можливість точно скопіювати вхід на виході. Зазвичай їх обмежують в розмірності коду (він менше, ніж розмірність сигналу) або штрафують за активації в коді. Вхідний сигнал відновлюється з помилками через втрати при кодуванні, але, щоб їх мінімізувати, мережа змушена вчитися відбирати найбільш важливі ознаки[24].

Реконструкція вхідного зображення автокодувальником у цьому методі проводиться за допомогою прогнозування. Проводиться індивідуальне тестування зображення, і вихід не є копією вхідного, але подібним до вхідного зображення. У цій складеній архітектурі (рисунок 2.1) нейронної мережі рівень коду має невеликий розмір порівняно з вхідною інформацією.

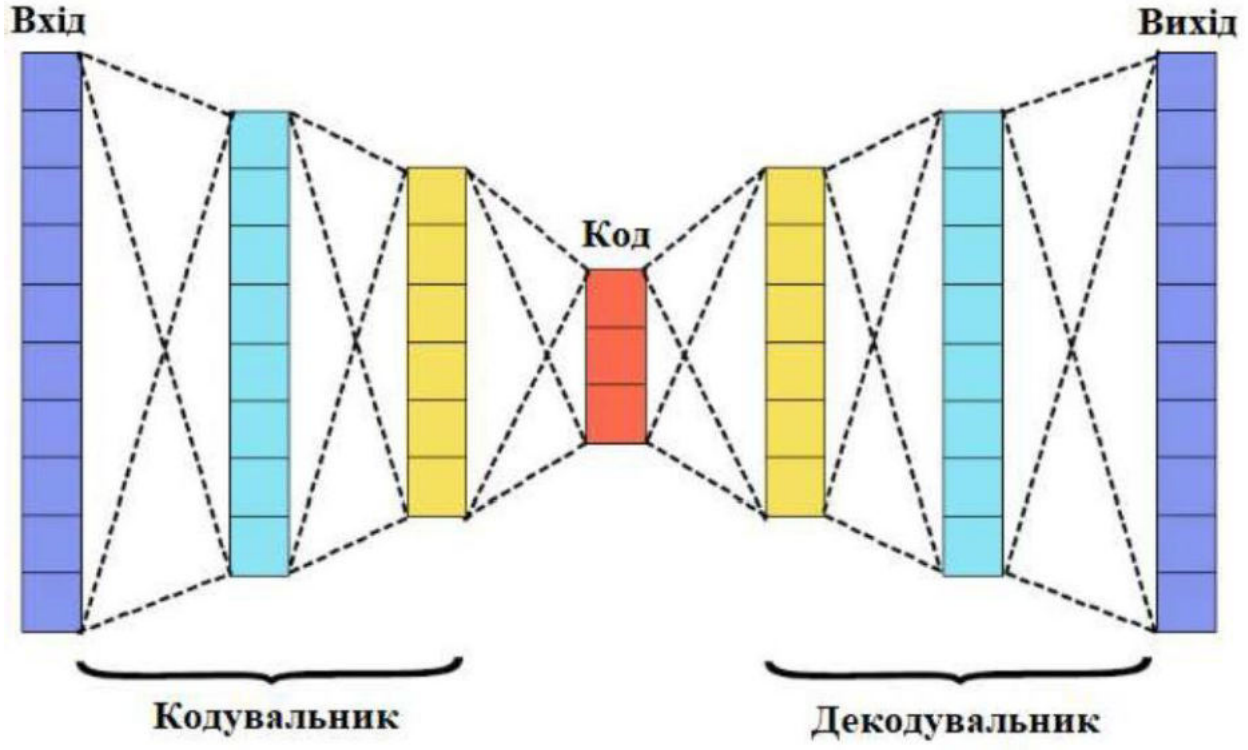


Рисунок 2.1 – Архітектура мережі

Згорткові автокодувальники (Convolutional AutoEncoders, CAE) - це архітектура згорткових нейронних мереж (CNN). Головна відмінність загальної інтерпретації CNN від CAE полягає в тому, що перші навчаються в кінці кінців до вивчення фільтрів та комбінування особливостей з метою класифікації їх вхідних даних. Насправді CNN зазвичай називають алгоритмами навчання з вчителем. Останні, натомість, навчаються лише для вивчення фільтрів, здатних витягувати функції, які можна використовувати для відновлення введення. Замість власноруч налаштованих згорткових фільтрів модель може вивчити оптимальні фільтри, що мінімізують помилку відновлення. Потім ці фільтри можна використовувати в будь-якому іншому завданні комп'ютерного зору. CAE є найсучаснішим інструментом для безконтрольного навчання згорткових фільтрів. Після засвоєння цих фільтрів їх можна застосувати до будь-яких вхідних даних з метої отримання певної функції. Тоді ці функції можна використовувати для виконання будь-якої задачі, яка вимагає компактного подання вхідних даних, наприклад класифікації[25].

CAE, завдяки їх згортковій природі, добре масштабують великі розміри зображень реалістичного розміру, оскільки кількість параметрів, необхідних для створення карти активації, завжди однакове, незалежно від розміру вводу. Тому CAE - це екстрактори функцій загального призначення, відмінні від AE, які повністю ігнорують 2D структуру зображення. Фактично, в AE зображення має бути розкручене в один вектор, а мережа повинна бути побудована за обмеженням кількості входів. Іншими словами, AE вводять надмірність параметрів, змушуючи кожну функцію бути глобальною (тобто охоплювати все зорове поле), тоді як CAE – ні.

Знешумлювальний автокодувальник (Denoising AutoEncoder, DAE) є розширенням звичайного автокодувальника і являє собою стохастичну його версію. Знешумлювальний автокодувальник намагається вирішити ризик функції ідентичності, випадково пошкоджуючи вхід (тобто вводячи шум), який автокодувальник повинен потім реконструювати або позначити[26].

Нижче, на рисунку 2.2 наведено ілюстрацію універсального автокодувальника.

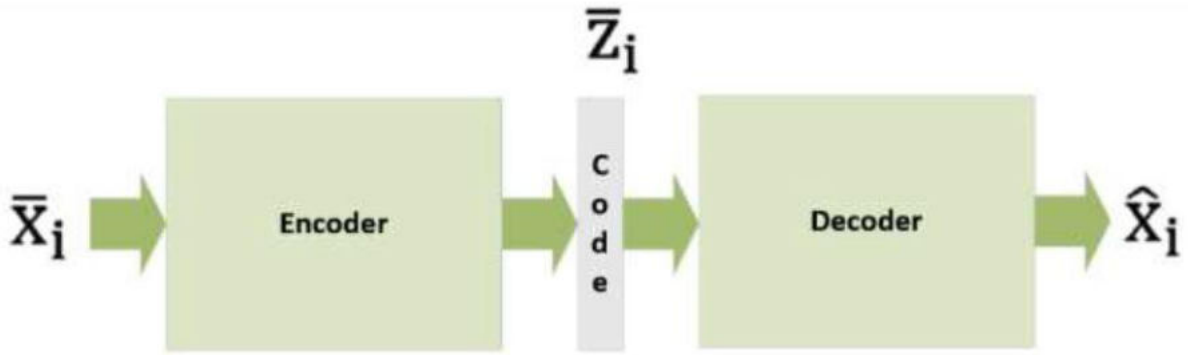


Рисунок 2.2 – Автокодувальник

Для p-мірного векторного коду параметризована функція, e( ) – є визначенням

$$\bar{z}_i = e(\bar{x}_i, \bar{\theta}_e) \text{ де } \bar{x}_i \in R^n \text{ та } \bar{z}_i \in R^p, \quad (2.1)$$

де $\bar{z}_i$ – код;
$\bar{x}_i$ – вхідні значення;
$\bar{\theta}_e$ – параметри

Аналогічно кодувальнику, декодувальник – це ще одна параметризована функція d( ):

$$\hat{x}_i = d(\bar{z}_i, \bar{\theta}_d) \text{ де } \hat{x}_i \in R^n \text{ та } \bar{z}_i R^p, \quad (2.2)$$

де $\hat{x}_i$ – відновлений вихід;
$\bar{\theta}_d$ – параметри декодувальника.

Таким чином, коли буде подано вхідні дані $\bar{x}_i$, повний автокодувальник, який за своєю суттю є послідовним виконнаням функцій кодувальника та декодувальника, забезпечить найкращу альтернативу як вихід:

$$\hat{x}_i = d(e(\bar{x}_i, \bar{\theta}_e), \bar{\theta}_d) = g(\bar{x}_i, \bar{\theta}), \quad (2.3)$$

Автокодувальник тренується на основі алгоритму зворотного поширення помилки. Фунція втрат часто базується на функції середньої квадратичної помилки:

$$C(X, \hat{X}, \bar{\theta}) = \frac{1}{m} \sum_i (\bar{x}_i - g(\bar{x}_i, \bar{\theta} \quad (2.4)$$

Звичайний та знешумлюючі автокодувальники за своєю суттю це одне й теж саме. Але важливо розуміти, що задача знешумлення потребує більш складної архітектури нейронної мережі, оскільки вона має відновити оригінальні зразки, з урахуванням пошкодженого вводу та кількість і розмір шарів можуть бути більшими, ніж для звичайного.

Звичайно, враховуючи складність, неможливо чітко зрозуміти без тестування, яку конкретні параметри обрати для мереж. Тому в даному випадку раціонально почати з меншої за розмірами моделі та збільшувати кількість шарів і їх параметри до тих пір, поки функція втрат не досягне відповідного значення. Для генерації зашумлених зображень (рисунок 2.3) можна використовувати припущення, що шум є Гауссовим, тобто розподіленим за нормальним законом.

Рисунок 2.3 – Результат додавання шум

Для оцінки якості роботи знешумлюючого автокодувальника було вирішено провести тестування його роботи. Для цього був реалізован прототип.

Перед регресійним аналізом було проведено тестування знешумлюючого автокодувальника для фільтрації зображень та інших нелінійних популярний лінійних фільтрів (нелокальне середнє, медіанне та курвлет перетворення) реалізованих на мові програмування. Результати тестів занесені до таблиці 2.1. Для оцінки якості роботи було вирішено використати пікове співвідношення сигналу до шуму (PSNR). Вимірюється у логарифмічній шкалі в дицебелах Це є співвідношенням між значенням максимально можливого значення сигналу та потужністю шуму, що спотворює значення сигналу.

$$PSNR = 10\log_{10}\left(\frac{MAX^2}{MSE}\right) = 20\log_{10}\left(\frac{MAX}{\sqrt{MSE}}\right), \quad (2.5)$$

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}|I(i,j) - K(i,j)|^2, \quad (2.6)$$

де MAX – максимальне значення пікселя;
середньоквадратична помилка моделі
Для регресійного аналізу обрано статистичний пакет SPSS[27].

Таблиця 2.1 – Результати тестів

ПОСШ	Розмір зображення, тис. пікс.	Алгоритм	Зашумленість
27,80	65,54	1,00	15,00
31,37	262,14	1,00	4,50
28,93	90,00	1,00	34,00
29,63	512,00	1,00	8,00
27,36	65,54	2,00	14,00
30,72	262,14	2,00	9,50
28,57	90,00	2,00	24,00
28,93	512,00	2,00	18,30
28,05	65,54	3,00	18,00

Продовження таблиці 2.1

ПОСШ	Розмір зображення, тис. пікс.	Алгоритм	Зашумленість
30,21	262,14	3,00	9,50
28,06	90,00	3,00	24,00
28,40	512,00	3,00	88,00
24,41	65,54	4,00	15,00
27,84	262,14	4,00	3,50
25,82	90,00	4,00	13,50
24,19	512,00	4,00	14,00

Оскільки не всі колонки таблиці мають цифрові значення, перед початком проведення аналізу таблицю потрібно привести до коректного стану(оцифрувати).

У графі «Алгоритм» використані позначення:

1. Нейронна мережа типу автокодувальник;
2. Курвлет перетворення;
3. Нелокальне середнє;
4. Вейвлет-перетворення.

Таблиця 2.2 – Результати розрахунків

	Посш	Размер_тыс_пикс	Алгоритм	Зашумленность
1	27,80	65,54	1,00	15,00
2	31,37	262,14	1,00	9,50
3	28,93	90,00	1,00	24,00
4	29,63	512,00	1,00	14,00
5	27,36	65,54	2,00	15,00
6	30,72	262,14	2,00	9,50
7	28,57	90,00	2,00	24,00
8	28,93	512,00	2,00	14,00
9	28,05	65,54	3,00	15,00
10	30,21	262,14	3,00	9,50
11	28,06	90,00	3,00	24,00
12	28,40	512,00	3,00	14,00
13	24,41	65,54	4,00	15,00
14	27,84	262,14	4,00	3,50
15	25,82	90,00	4,00	24,00
16	24,19	512,00	4,00	14,00

Для регресійного аналізу в якості залежної зміни обрано час аналізу.

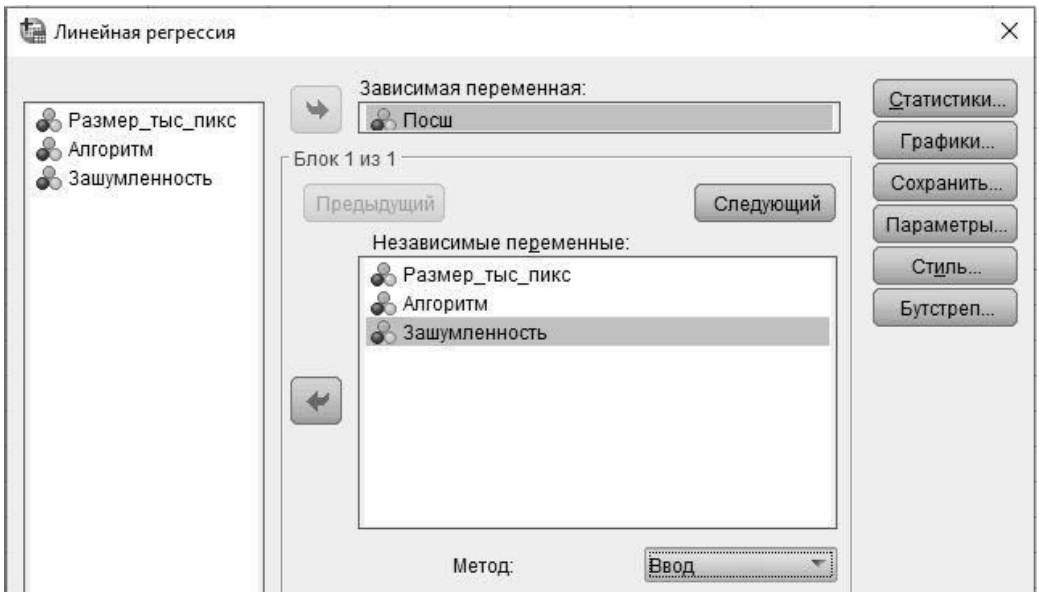


Рисунок 2.4 – Параметри лінійної регресії

Введенные/удаленные переменные[a]

Модель	Введенные переменные	Удаленные переменные	Метод
1	Зашумленность, Алгоритм, Размер_тыс_пикс[b]	.	Enter

a. Зависимая переменная: Посш

b. Все требуемые переменные введены.

Сводка для модели

Модель	R	R-квадрат	Скорректированный R-квадрат	Стандартная ошибка оценки
1	,948[a]	,899	,874	2,40310

a. Предикторы: (константа), Зашумленность, Алгоритм, Размер_тыс_пикс

Коэффициенты[a]

Модель		Нестандартизованные коэффициенты B	Нестандартизованные коэффициенты Стандартная ошибка	Стандартизованные коэффициенты Бета	t	Значимость
1	(Константа)	36,198	1,711		21,151	,000
	Размер_тыс_пикс	-,003	,003	-,090	-,953	,360
	Алгоритм	-2,418	,538	-,412	-4,496	,001
	Зашумленность	-,278	,032	-,813	-8,627	,000

a. Зависимая переменная: Посш

Рисунок 2.5 – Лінійна регресія

Отримана модель:

$$У = 36{,}198 – 0{,}003 * X1 - 2{,}418 * X2 - 0{,}278 * X3$$

Значення X відповідають:

X1– Кількість пікселів;

X2 – Алгоритм;

X3 – Яскравість.

Оцінка якості моделі

Коефіцієнт детермінації: R2= 0,948.

Показує частку варіації результативного признаку під впливом досліджуваних факторів. Отже, близько 95% варіації залежної змінної враховано і обумовлено в моделі впливом включених факторів.

Коефіцієнт множинної кореляції R = 0,899.

Показує тісноту зв'язку між залежною змінною У з усіма включеними в модель факторами.

Отже, рівняння регресії слід вважати адекватним, модель є значимою.

Для оптимізації отриманої багатофакторної регресивної моделі був використаний метод «Forward».

Введенные/удаленные переменные[a]

Модель	Введенные переменные	Удаленные переменные	Метод
1	Зашумленность		Пошаговый (критерий: Вероятность F для включения <= ,050, Вероятность F для исключения >= ,100).
2	Алгоритм		Пошаговый (критерий: Вероятность F для включения <= ,050, Вероятность F для исключения >= ,100).

Сводка для модели

Модель	R	R-квадрат	Скорректированный R-квадрат	Стандартная ошибка оценки
1	,850[a]	,723	,703	3,69177
2	,944[b]	,892	,875	2,39453

a. Предикторы: (константа), Зашумленность

b. Предикторы: (константа), Зашумленность, Алгоритм

Коэффициенты[a]

Модель		Нестандартизованные коэффициенты B	Нестандартизованные коэффициенты Стандартная ошибка	Стандартизованные коэффициенты Бета	t	Значимость
1	(Константа)	29,635	1,317		22,497	,000
	Зашумленность	-,290	,048	-,850	-6,042	,000
2	(Константа)	35,558	1,569		22,669	,000
	Зашумленность	-,285	,031	-,834	-9,130	,000
	Алгоритм	-2,413	,536	-,411	-4,503	,001

a. Зависимая переменная: Посш

Исключенные переменные[a]

Модель		Бета-включения	t	Значимость	Частная корреляция	Статистика коллинеарности Допуск
1	Размер_тыс_пикс	-,086[b]	-,578	,573	-,159	,946
	Алгоритм	-,411[b]	-4,503	,001	-,781	,999
2	Размер_тыс_пикс	-,090[c]	-,953	,360	-,265	,946

a. Зависимая переменная: Посш

b. Предикторы в модели: (константа), Зашумленность

c. Предикторы в модели: (константа), Зашумленность, Алгоритм

Рисунок 2.6 – Оптимізація лінійної регресії методом «Forward»

Отримана модель: У = 35,558 - 0,285 * X3 - 2,413 * X2

Оцінка якості моделі

В результаті оптимізації було з’ясовано, що найменший вплив на залежну зміну(ПОСШ) має критерій Размер_тыс_пикс.

Коефіцієнт детермінації: R^2= 0,892.

Коефіцієнт множинної кореляції R = 0,944.

Аналіз візуальних і чисельних результатів показує, що для зазначених тестових зображень і ступенів зашумлення алгоритм фільтрації на основі автоенкодера показує найкращі результати порівняно з розглянутими підходами. І наступне його використання є оправданим кроком.

2.2 Модифікація нейронної мережі

2.2.1 Агрегувальний шар

Звичайна нейронна мережа побудована у вигляді послідовних згорткових та агрегувальних шарів. Агрегувальний шар існує для зменшення значення активації у кожній просторово-локальній області в розглянутому каналі. У сучасних системах візуального розпізнавання операції субдискритизації (агрегування) відіграють певну роль у створенні ознак, які є більш стійкими до впливу варіацій даних, зберігаючи важливі елементи. Конкретний вибір між середнім максимальним об'єднанням є окремими питанням для багатьох архітектур нейронних мереж.

Було розроблено функцію, на меті якої стояло поєднання максимальної та середньої субдискретизації, а саме навчання та чутливість. Головною стратегією для цього є "невідповідність" характеристикам регіону, який об'єднується; процес навчання в цій стратегії призведе до ефективної операції об’єднання, яка є певним специфічним поєднанням максимуму та середнього[28].

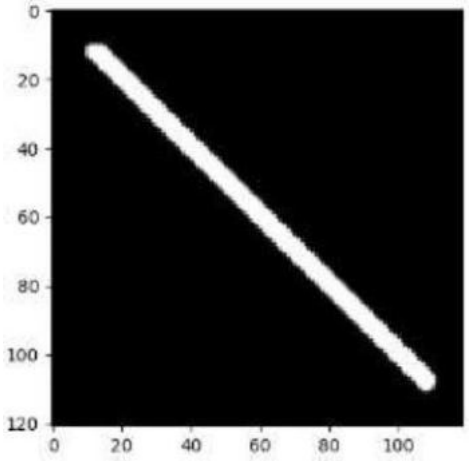


Рисунок 2.7 – Вхідне зображення

Звичайними операціями для агрегувального шару є максимальне $f_{max}(x)$ та середнє $f_{avg}(x)$ значення (рисунок 2.8 та 2.9 відповідно). В даний час функція максимального значення використовується завжди для агрегувального шару у сучасних згорткових мережах.

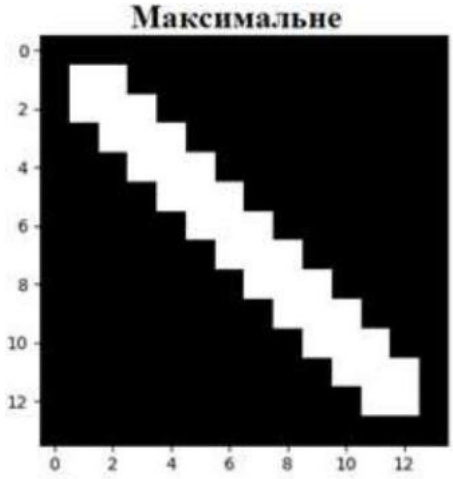


Рисунок 2.8– Зображення після агрегації з max

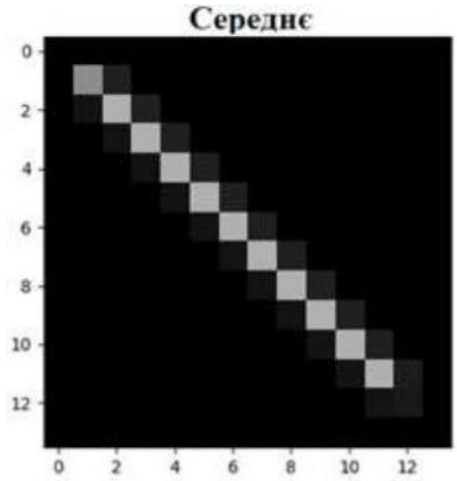


Рисунок 2.9 – Зображення після агрегації з avg

Головна ідея змішаного агрегування полягає в тому, щоб отримати більше інформації без збільшення розміру вихідного масиву. Це засновано на поєднанні середнього та максимального у певній пропорції, що дозволяє отримати більше інформації подібно природному узагальненню. Це дозволяє вплинути на розпізнавання зображень і збільшити інваріантність загальних перетворень зображень (включаючи обертання, масштабування, спотворення).

Загальна формула для змішаної агрегації буде мати наступний вигляд:

$$f_{mix}(x) = a \times f_{max}(x) + (1 - a) \times f_{avg}(x), \quad (2.7)$$

де $a \in [0,1]$.

За своєю природую a дозволяє обирати пропорцію змішування між максимальним та середнім.

Візуалізація роботи

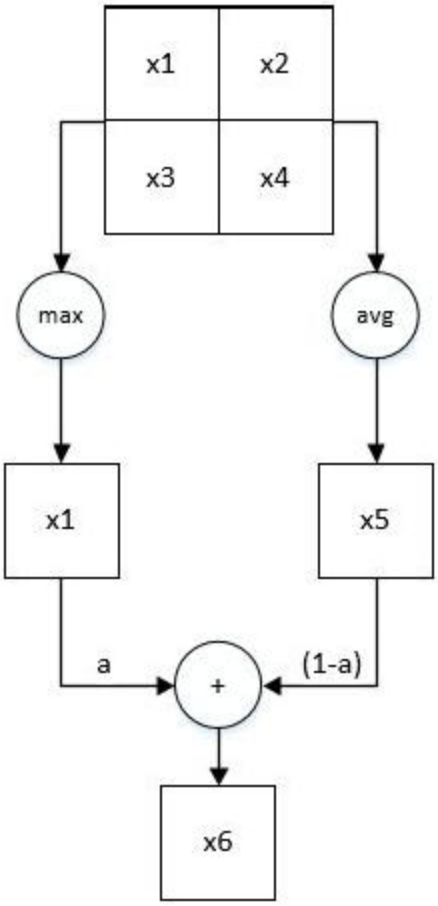


Рисунок 2.10 – Загальна схема змішанного агрегування (x1 – максимальне)

Наприклад, якщо $a = 0$, то $f_{mix}(x) = f_{avg}(x)$ та якщо $a = 1$, то $f_{mix}(x) = f_{max}(x)$.

Також слід зазначити, що головна складність полягає в обиранні параметру a. Оскільки його потрібно обирати окремо для кожного агрегувального шару, це ускладнюю задачу. Вплив a не результат агрегування наведено на рисунках 2.11-2.13.

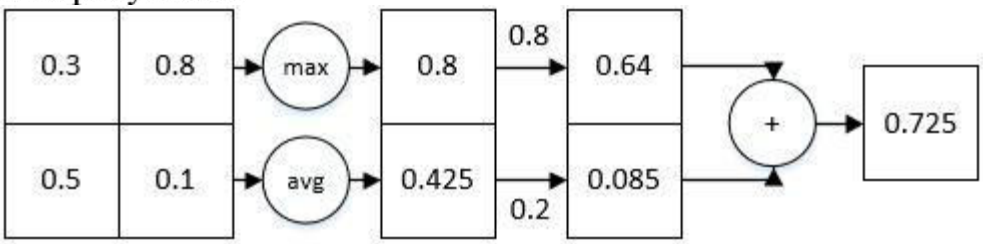


Рисунок 2.11 – Змішане агрегування при a = 0.8

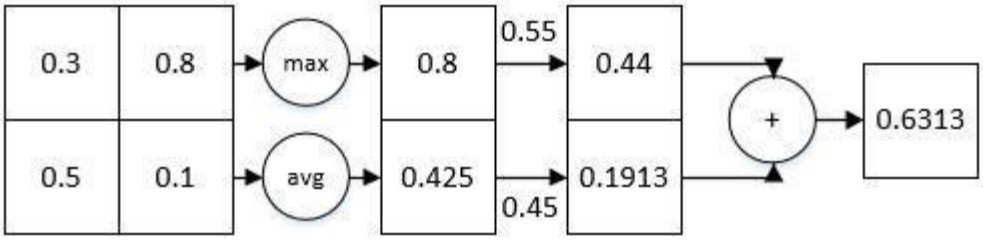


Рисунок 2.12 – Результати розрахунків, де a = 0.55

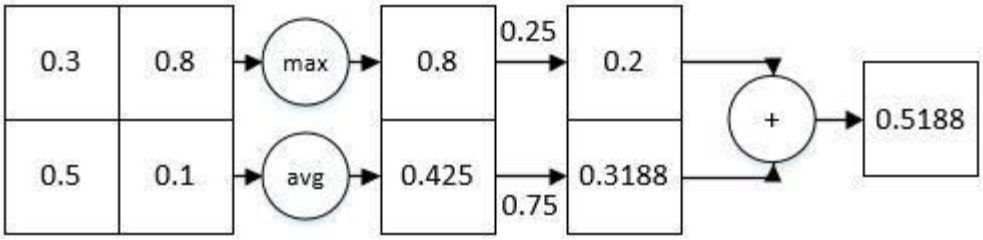


Рисунок 2.13 – Змішане агрегування при a = 0.25

Вибір конкретного значення параметру для змішування доволі сильно впливає на кінцевий результат. Необхідно дотримуватися однієї стратегії при виборі значення, щоб зменшити вплив випадковості.

Оскільки в данній роботі необхідно створити декілька нейронних мереж, можливо застосувати змішану агрегацію з різними параметрами в залежності від цілей (мережа-класифікатор та автокодувальники).

Для знешумлюючого автокодувальника запропоновано використання нормального розподілу (розподілу Гауса) для зміни значення параметру змішування в межах стандартного відхилення.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{2.8}$$

де μ – математичне сподівання (середнє значення);

σ – середньоквадратичне відхилення().

Приклад нормального розподілу наведено на рисунку 2.14.

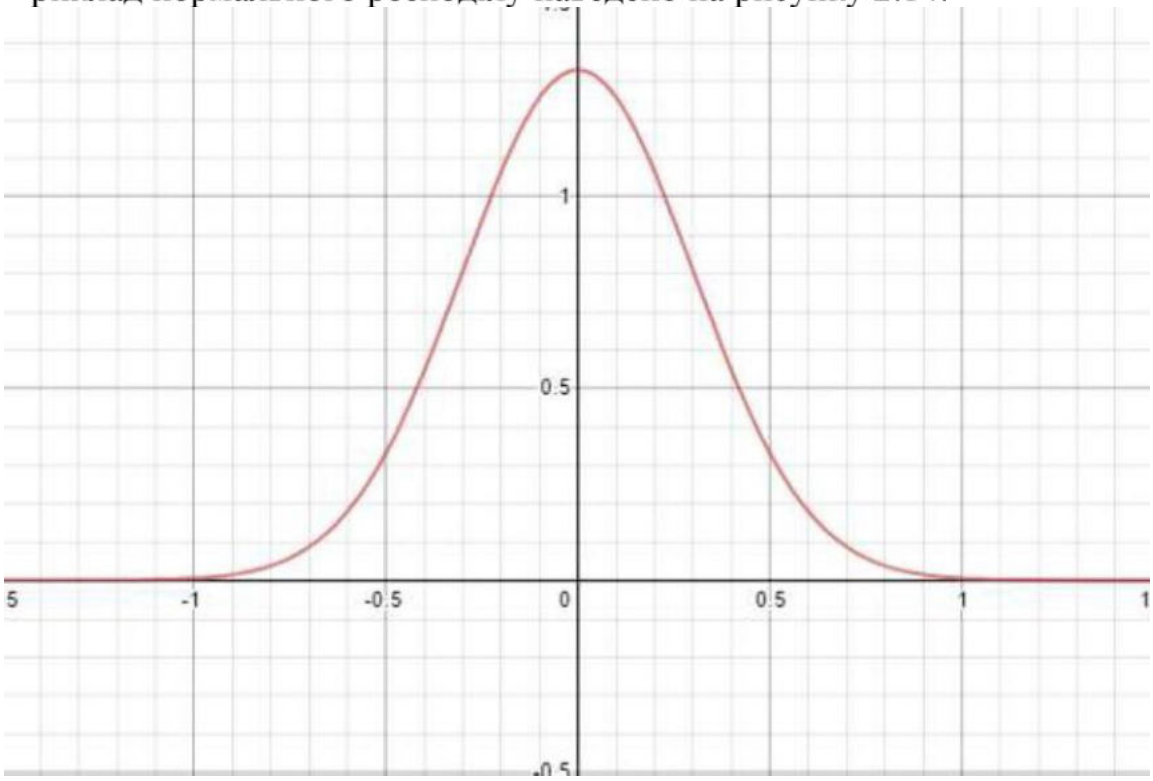


Рисунок 2.14 – Нормальний розподіл при μ = 0 та σ = 1

Отже, основна ідея удосконалення змішаного агрегування полягає в використанны нормального розподілу для збільшення інваріативності та показника точності.

Але важливо зазначити, що випадковість відносно початкової величини параметра не має бути великим числом. Наприклад для a = 0.7, $a \in [0.65{,}075]$

Таблиця 2.3 Значення функції втрат нейронних мереж при різних агрегувальних функціях з агрегувальним шаром a = 0.85.

Номер епохи	Середнє	Максимальне	Змішане	Змішане Випадкове
1	0.0385	0.0368	0.0356	0.0331
2	0.0241	0.0209	0.0203	0.0204
3	0.0237	0.0202	0.0196	0.0194
4	0.0228	0.0195	0.0194	0.0193

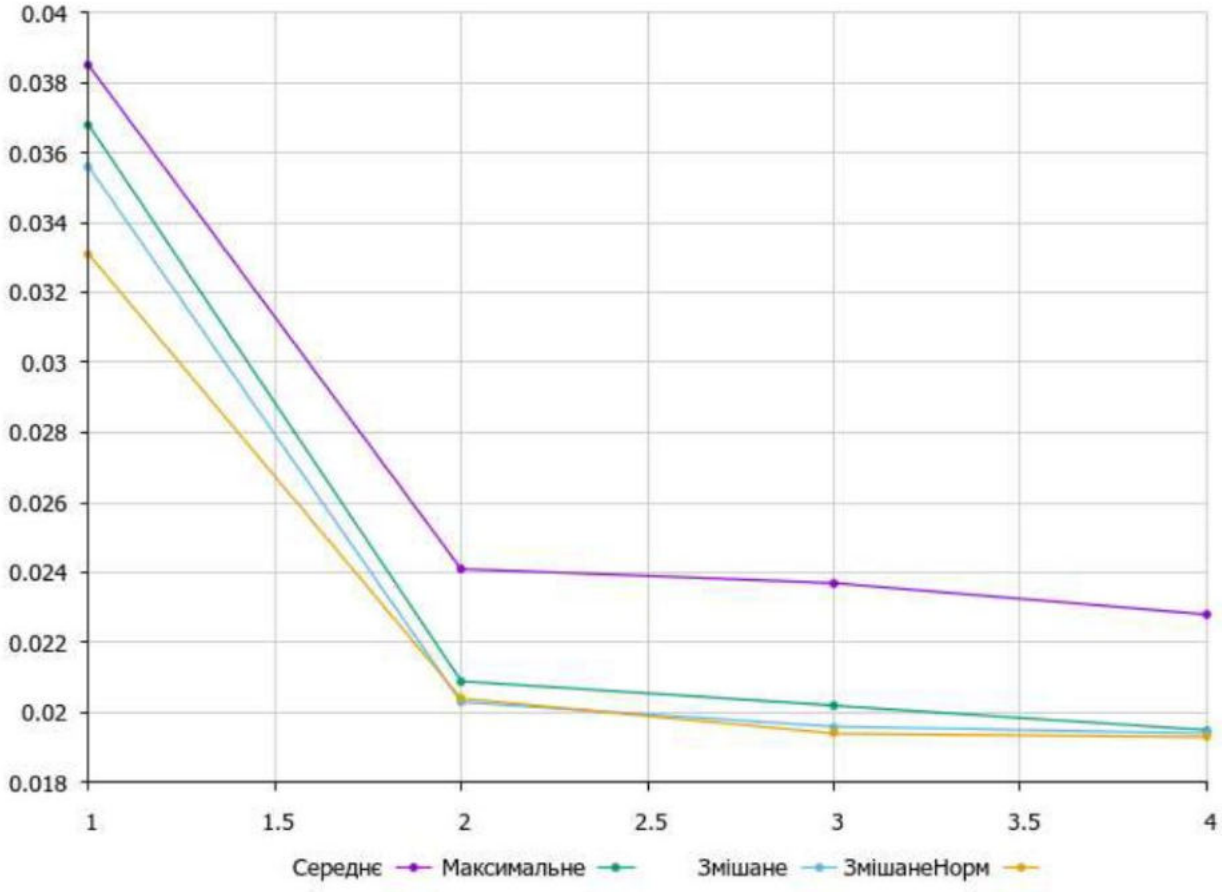


Рисунок 2.15 – Графіки значень функції втрат роботи нейронних мереж

Отже, із результатів тестів можна зробити висновок, що модифікований в даному підрозділі агрегувальний шар на основі поєднання максимального та середнього майже не має виграшу у задачі знешумлення на випадкових вхідних данних.

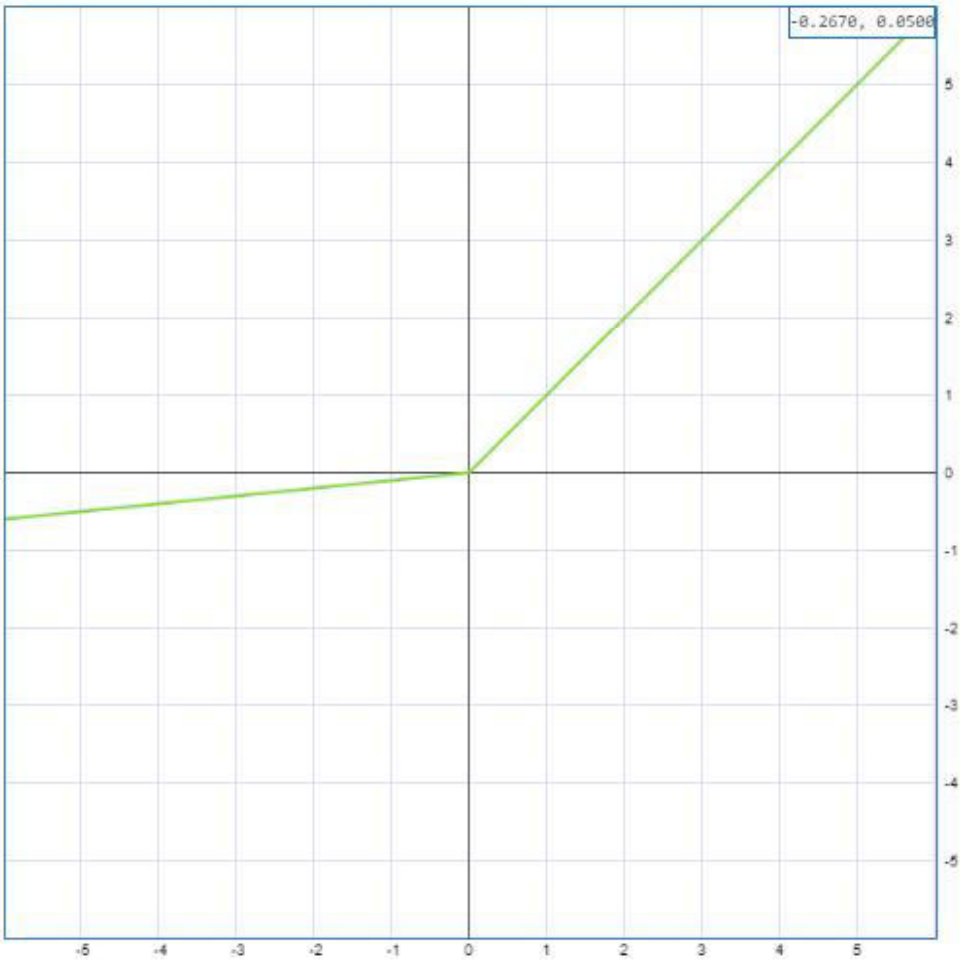


Рисунок 2.16 – Графік Leaky ReLU

Функція Leaky ReLU намагається врахувати той факт, що 0 не сильно підходить для навчання мережі. Але в даному випадку це не сильно вирішує проблему, але й додає нових, оскільки в певних випадках від'ємні значення на виході нейрона можут стати занадто великими і це також може негативно вплинути на результат навчання.

В результаті чого, Brain Team Google запропонувала нову активаційну фукнцію (рисунок 2.17) під назвою Swish, яка визначається наступним чином:

$$f(x) = x \times \sigma(x) = x \times \frac{1}{1 + e^{-x}}, \qquad (2.12)$$

Їх експерименти показують, що Swish працює краще, ніж ReLU, на більш глибоких моделях у ряді складних наборів даних. Наприклад, проста заміна ReLU на модулі Swish покращує точність класифікації Top-1 у ImageNet на 0,9% та для Inception-ResNet-v2 на 0,6%. Простота Swish та його схожість з ReLU дозволяє легко замінити активаційну функцію ReLU на Swish в будь-якім шарі нейромережі [30].

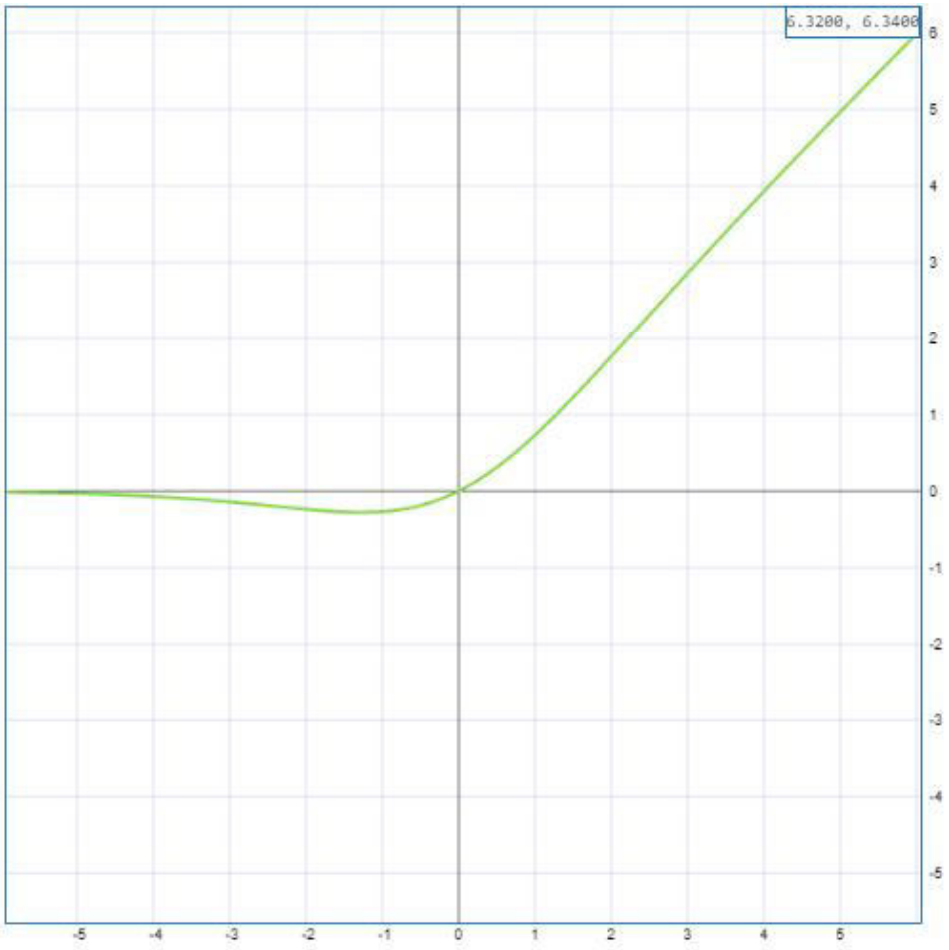


Рисунок 2.17 – Графік Swish

На питання, чому Swish може бути кращим за ReLU автори оригінальної роботи роблять різні спостереження, які намагаються пояснити таку поведінку:

- Фукнція обмежена знизу. Таким чином, Swish отримує перевагу від невеликих від'ємних чисел. Дуже негативні ваги просто дорівнюють нулю;
- Функція не обмежена вище. Це означає, що для дуже великих значень виходи не насичуються до максимального значення (тобто до 1 для всіх нейронів). На думку авторів статті Swish, саме це виділяє ReLU від більш традиційних функцій активації;
- Малі від'ємні значення дорівнюють нулю в ReLU. Однак ці негативні значення все ще можуть бути актуальними для навчання фільтрів, тощо, що лежать в основі даних, тоді як великі негативні значення можуть бути нульовими.

Для Swish також існує модифікація, коли до сигмоїди додається параметр β:

$$f(x) = x \times \sigma(\beta x) = x \times \frac{1}{1 + e^{-\beta x}}, \qquad (2.13)$$

Тут β - параметр, який необхідно налаштувати. β повинен відрізнятися від 0, інакше функція Swish стає лінійною функцією. Якщо β наближається до ∞, то функція є збіжною.

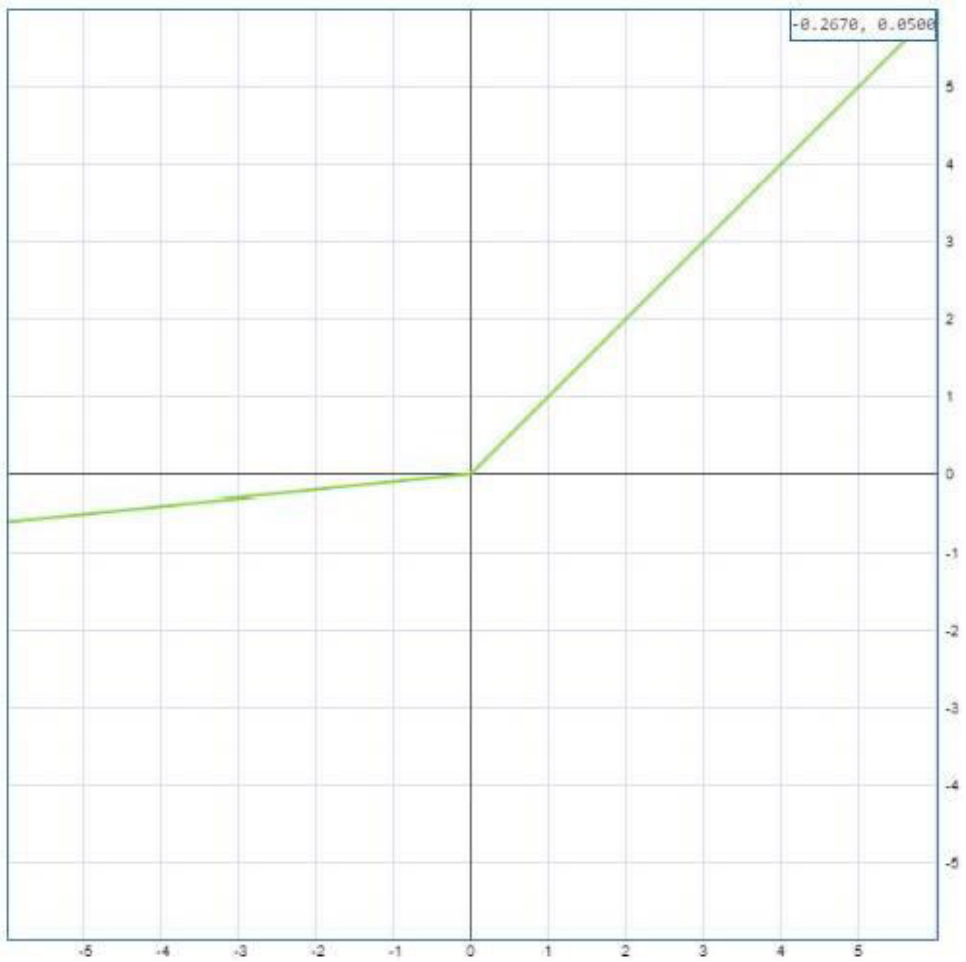


Рисунок 2.16 – Графік Leaky ReLU

Функція Leaky ReLU намагається врахувати той факт, що 0 не сильно підходить для навчання мережі. Але в даному випадку це не сильно вирішує проблему, але й додає нових, оскільки в певних випадках від'ємні значення на виході нейрона можут стати занадто великими і це також може негативно вплинути на результат навчання.

В результаті чого, Brain Team Google запропонувала нову активаційну фукнцію (рисунок 2.17) під назвою Swish, яка визначається наступним чином:

$$f(x) = x \times \sigma(x) = x \times \frac{1}{1 + e^{-x}}, \qquad (2.12)$$

Їх експерименти показують, що Swish працює краще, ніж ReLU, на більш глибоких моделях у ряді складних наборів даних. Наприклад, проста заміна ReLU на модулі Swish покращує точність класифікації Top-1 у ImageNet на 0,9% та для Inception-ResNet-v2 на 0,6%. Простота Swish та його схожість з ReLU дозволяє легко замінити активаційну функцію ReLU на Swish в будь- [30].

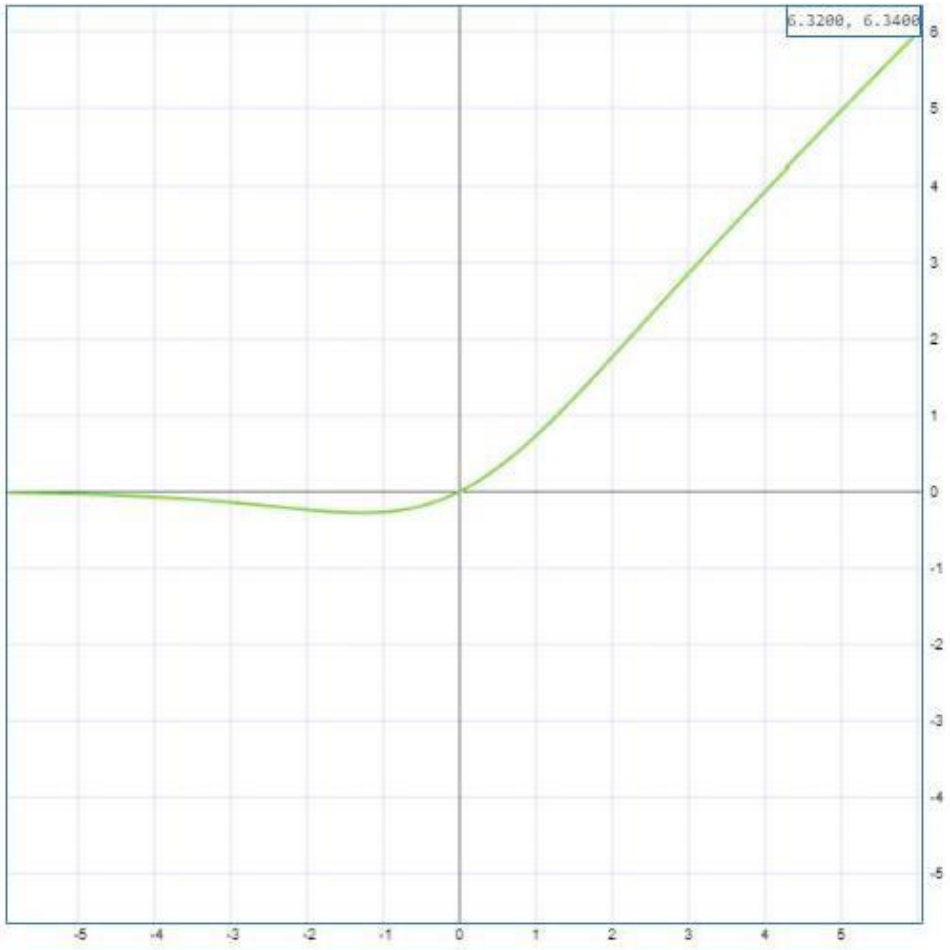


Рисунок 2.17 – Графік Swish

На питання, чому Swish може бути кращим за ReLU автори оригінальної роботи роблять різні спостереження, які намагаються пояснити таку поведінку:

- Фукнція обмежена знизу. Таким чином, Swish отримує перевагу від невеликих від'ємних чисел. Дуже негативні ваги просто дорівнюють нулю;
- Функція не обмежена вище. Це означає, що для дуже великих значень виходи не насичуються до максимального значення (тобто до 1 для всіх нейронів). На думку авторів статті Swish, саме це виділяє ReLU від більш традиційних функцій активації;
- Малі від'ємні значення дорівнюють нулю в ReLU. Однак ці негативні значення все ще можуть бути актуальними для навчання фільтрів, тощо, що лежать в основі даних, тоді як великі негативні значення можуть бути нульовими.

Для Swish також існує модифікація, коли до сигмоїди додається параметр β:

$$f(x) = x \times \sigma(\beta x) = x \times \frac{1}{1 + e^{-\beta x}}, \qquad (2.13)$$

Тут β - параметр, який необхідно налаштувати. β повинен відрізнятися від 0, інакше функція Swish стає лінійною функцією. Якщо β наближається до ∞, то функція

Swish β вважався рівним 1. Нижче наведено вплив на β на результуючий графік.

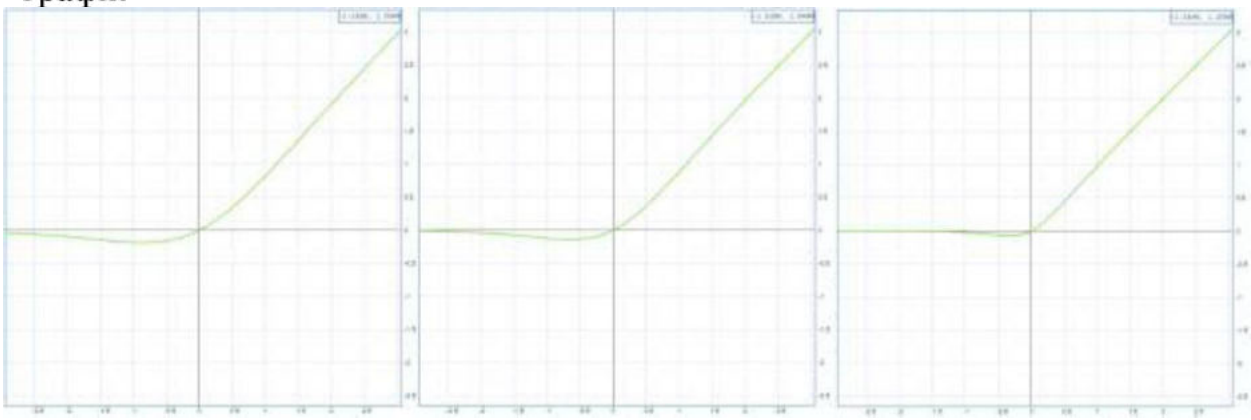


Рисунок 2.18 – Графіки Swish при = 1.5, = 2 та $\beta = 4$

В навчанні нейронних мереж використовується похідна від функції активації, виведемо її наступним чином для Swish.

Оскільки рівняння містить дві диференційовні функції, можна скористатися правилом добутку:

$$(f \times g)' = f' \times g + f \times g', \tag{2.14}$$

Спростимо вигляд $f(x)$ замінивши $\frac{}{+e^{-x}}$ на позначення сигмоїди $\sigma(x)$:

$$f(x) = x \times \frac{1}{1+e^{-x}} = x \times \sigma(x), \tag{2.15}$$

Обчислимо значення похідної $f(x)$:

$$f'(x) = x' \times \sigma(x) + x \times \sigma'(x) =$$
$$= \sigma(x) + x \times \sigma(x) \times (1 - \sigma(x =$$
$$= \sigma(x) + x \times \sigma(x) - x \times (\sigma(x^{2}$$
$$x \times \sigma(x) + \sigma(x) - x \times (\sigma(x^{2} =$$
$$= f(x) + \sigma(x) - x \times (\sigma(x^{2}$$
$$f(x) + \sigma(x) \times (1 - x \times \sigma(x =$$
$$= f(x) + \sigma(x) \times (1 - f(x$$

$$f'(x) = x \times \frac{1}{1+e^{-x}} + \frac{1}{1+e^{-x}} \times \left(1 - x \times \frac{1}{1+e^{-x}}\right), \tag{2.16}$$

Спростимо вигляд функції розкриваючи дужки:

$$f'(x) = \frac{e^{-x}(x+1)+1}{\left(\frac{}{1+e^{-x}}\right)^{2}}, \tag{2.15}$$

Головним недоліком Swish є факт, що обчислення функції має більшу вартість, ніж ReLU, як для прямогу обчислення, так і для зворотного.

З огляду на все вищенасказане було вирішено використовувати в нейронних мережах окрім ReLU також і Swish, для отримання більш кращого результату.

Висновки за розділом

Отже, в даному розділі було наведено особливості та архітектуру автокодувальників, можливих сфер їх застосувань. Проведено регресійний аналіз, який показав, що найбільший влив на пікове відношення шуму до сигналу між пошкодженим та відновленим зображенням мережею мають такі фактори як, розмір вхідного зображення, відсоток зашумленості та алгоритм фільтрації. Також було досліджено новітню функцію активації Swish та змішане агрегування. Запропоновано використання змішаного агрегувального шару з долею випадковості.

3 РОЗРОБКА ДОДАТКУ ДЛЯ ФІЛЬТРАЦІЇ ТА РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ

3.1 Концепт майбутньої програми

Головна ідея програми полягає у послідовному використанні різних згорткових мереж (рисунок 3.1), які поліпшать якість розпізнавання вхідного зображення.

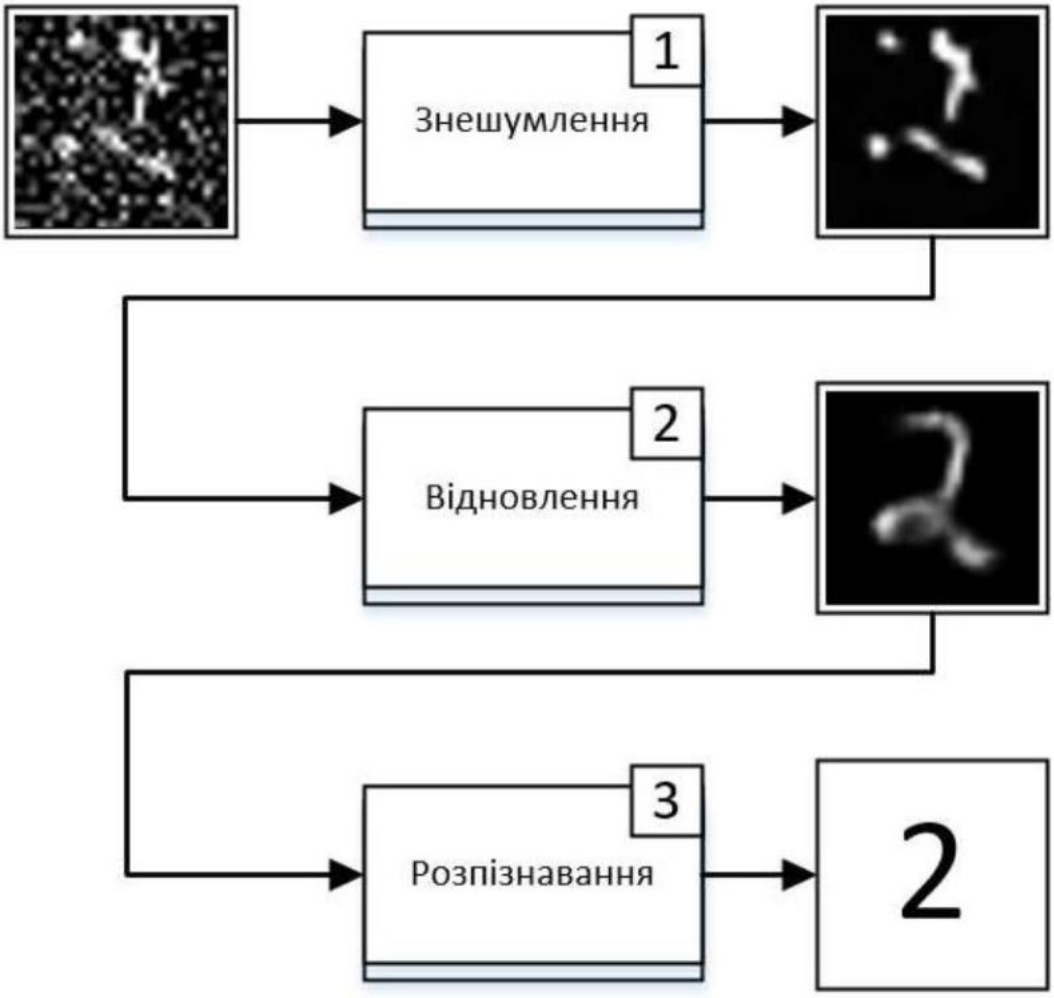


Рисунок 3.1 – Схема послідовної обробки зображення

Послідовність виконання обробки зображення:

1. Знешумлення;
2. Відновлення;
3. Розпізнавання.

Важливо зазначати, що кроки 1 та 2 не є необхідними для розпізнавання зображень, оскільки це може займати лише зайвий час. З огляду на це, програму необхідно спроектувати таким чином, щоб кроки 1 та 2 могли бути застосованими в будь-якому порядку.

Головні вимоги, яким має відповідати створена програма:

1. Простота використання;
2. Модульність;
3. Прийнятна швидкість роботи;
4. Наочність результатів обробки зображень нейронними мережами.

Одним з основних алгоритмів є процес взаємодії програми з користувачем, для демонстрації якого була створена UML діаграма послідовності (рисунок 3.2). Вона відображає основні етапи роботи з додатком.

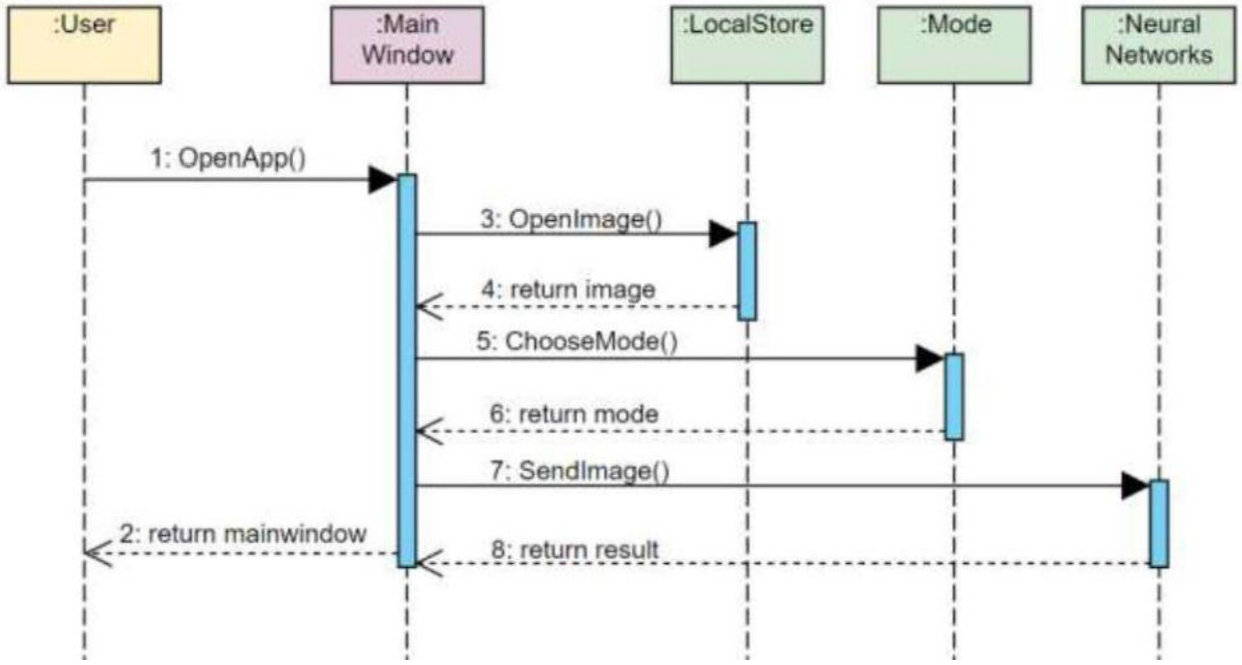


Рисунок 3.2 – UML діаграма послідовності роботи з додатком

На початку користувач відкриває додаток та переходить до головного екрана. Наступним кроком він може обрати режим обробки зображення. За замовчуванням буде ввімкнено розпізнавання зображень. Після необхідно обрати зображеня, які необхідно завантажити до програми. Після обирання режиму роботи додатку та вхідного зображення, необхідно натиснути необхідно кнопку та надати зображення на вхід нейронної мережі. Після обробки зображення, мережа надсилає значення свого виходу до головного екрана додатку.

Щодо наочності результатів обробки зображень, для цього необхідно створити діаграму, яка б показувала у процентному співвідношенні силу активації вихідних нейронів у мережі для розпізнавання зображень (рисунок 3.3).

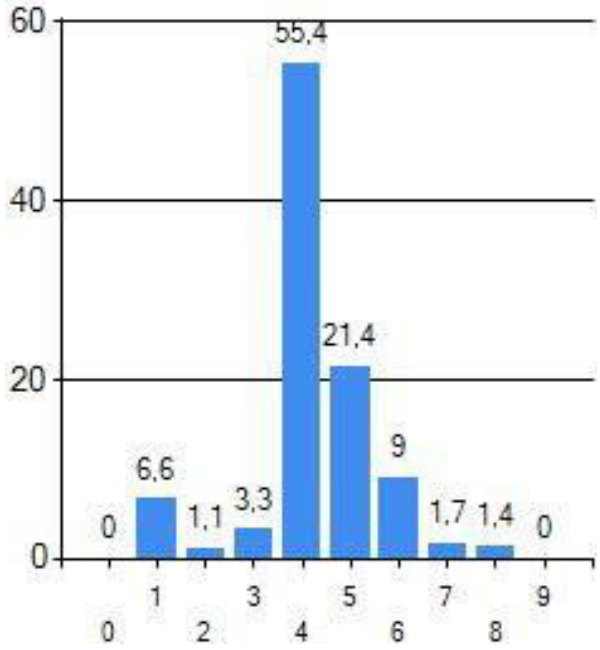


Рисунок 3.3 – Відповідь нейронної мережі після розпізнавання

Для вибору конкретних параметрів нейронних мереж, правильним кроком буде почати з менших за розмірами моделями та збільшувати кількість шарів (їх параметри до тих пір), поки функція втрат не досягне відповідного (задовільного) значення.

Щодо архітектур нейронних мереж, то взагалі немає правил з вибору числа згорткових шарів у мережі, проте точно відомо, що використання одного шару не призводить до покращених значень точності класифікації. Це пояснюються тим, що одного шару замало для вивчення складних ознак, які потім могли б бути вивчені іншими згортковими шарами для знаходження більш складних ознак.

Також необхідно приділити увагу вибору метода оптимізації нейронної мережі. Keras надає наступні методі оптимізації з налаштуваннями параметрів за замовчуванням:

1. SGD;
2. RMSprop;
3. Adagrad;
4. Adadelta;
5. Adam;
6. Adamax.

Вибір оптимізатора залежить від конкретної задачі та може змінюватися від багатьох параметрів. Мережі глибинного навчання, як правило, навчаються оптимізатором стохастичного градієнтного спуску. Але в даній роботі використовується Adam. Даний метод дозволяє встановити швидкість

переміщати ваги у напрямку градієнта. Якщо рівень навчання є низьким, то тренування є більш надійними, але оптимізація займе багато часу, тому що кроки до мінімальної функції втрат невеликі. Якщо рівень навчання високий,

навчання може не сходитися. Зміни ваги можуть бути настільки великими, що оптимізатор перевищує мінімум і може погіршити значення втрат.

3.2 Вибір фреймворку для машинного навчання

Для створення програмного додатку, який реалізує запропоновану математичну модель рішення задач, було використано мову програмування Python (версія 3.7.4). Вибір Python як мови програмування зумовлений тим, що це найбільш популярна мова програмування, під яку існує найбільша кількість фреймворків для машинного навчання та нейронних мереж. Інтерфейс користувача буде створено на основі Windows Forms, мови програмування C# у середовищі Visual Studio 2017. Також необхідні додаткові бібліотеки для математичної статистики та побудови графіків.

Важливими критеріями для вибору фреймворку є наявність можливості створення та навчання саме згорткових нейронних мереж. Фреймворкі, які не мають цієї можливості відповідно не будуть розглядатися. Також фреймворк має буди відкритим та мати зручну документацію, що спростить його використання.

Таблиця 3.1 – Порівняння фреймворків для глибинного навчання

Назва	Платформа	Мова для використання	OpenMP	CUDA
TensorFlow[31]	Linux, Mac OS X, Windows	Python, C/C++, Java, Go	Ні	Так
Caffe	Linux, Mac OS X, Windows	Python, MATLAB		Так
Theano	Linux, Mac OS X, Windows	Python		Так
PyTorch	Linux, Mac OS X, Windows, Android, iOS	PyTorch, Lua, LuaJIT, C, C++/OpenCL	Так	Так

Далі буде розглянуто більш детально з точки зору переваг та недоліків вищезазначених фреймворки.

TensorFlow

Переваги:

1. Для неї написано велику кількість посібників та документації;
   Вона пропонує потужні засоби моніторингу процесу навчання моделей і візуалізації (Tensorboard);
2. Підтримується великою спільнотою розробників і технічними компаніями;
3. Забезпечує обслуговування моделей;
4. Підтримується кросплатформеність;

5. TensorFlow Lite забезпечує вивід на пристрої з низькою затримкою для мобільних пристроїв;

Недоліки:

1. Програє по швидкості роботи в еталонних тестах;
2. Має більш високий вхідний поріг для початківців, ніж PyTorch або Keras;
3. Tensorflow досить низькорівнева і вимагає багато шаблонного коду, і режим «визначити і запустити» для Tensorflow значно ускладнює процес відладки.

Caffe

Переваги:

1. Пропонує попередньо навчені моделі для створення демонстраційних додатків;
2. Швидкий, масштабований і займає мало місця;
3. Він добре працює з іншими фреймворками, такими як PyTorch.

Недоліки:

1. Обмежена підтримка спільнотою.

Theano

Переваги:

1. Платформа відмінно оптимізована для роботи як з CPU, так і з GPU;
2. [illegible] інтенсивним використанням даних, але вимагає об'єднання з іншими бібліотеками.

Недоліки:

1. Для поточної версії Theano не запланований випуск оновлень і додавання функціоналу.

PyTorch

Переваги:

1. Завдяки архітектурі фреймворка, процес створення моделі досить простий і прозорий;
2. Режим за замовчуванням "define-by-run" - відсилання до традиційного програмування. Фреймворк підтримує популярні інструменти для дебага, такі як pdb, ipdb або дебагер PyCharm.
3. Він має багато попередньо навчених моделей і готових модульних частин, які легко комбінувати;

Недоліки:

1. Недостатня підтримка моделей;
2. Бракує інтерфейсів для моніторингу

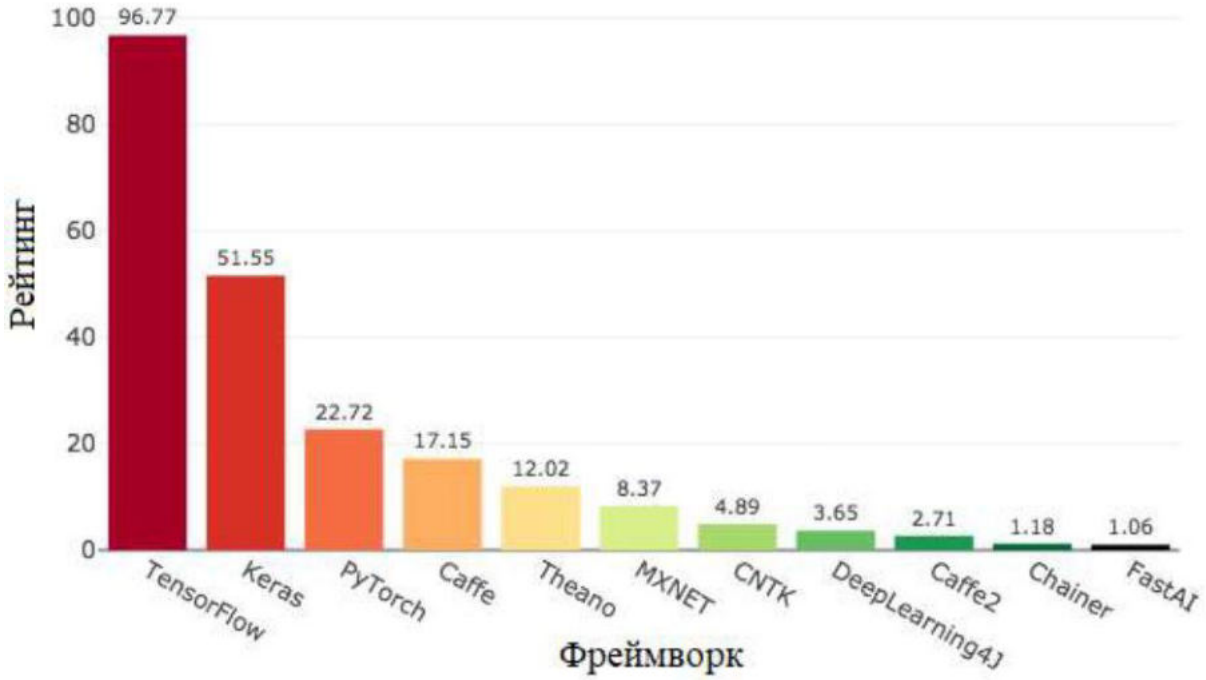


Рисунок 3.4 – Рейтинг інтересів й зацікавленості фреймворками інтернет-спільнотою у 2018-2019 роках

TensorFlow – безперечний чемпіон в суперважкій вазі. Він має найбільшу активність в GitHub, пошуки в Google, статті Medium, книги за статтями Amazon і ArXiv. У нього також є більшість розробників, що використовують це і перераховане в більшості описів роботи онлайн. TensorFlow підтримується Google. TensorFlow є однією з кращих відкритих бібліотек для чисельних обчислень. Вона просто зобов'язана бути хорошою, оскільки навіть такі гіганти як DeepMind, Uber, AirBnB або Dropbox вибрали цей фреймворк для своїх потреб.

TensorFlow добре підходить для складних проектів, таких як створення багатошарових нейронних мереж. Вона використовується для розпізнавання голосу або картинок і додатків для роботи з текстом, таких як Google Translate, наприклад.

Оскільки в результаті роботи необхідно створити декілька нейронних мереж для різних цілей, то для більш зручного створення та налаштування нейронних мереж було вирішено використовувати Keras. Keras – API

ад TensorFlow, CNTK або Theano. Він був розроблений з акцентом на швидке експериментування. Можливість переходити від ідеї до результату з найменшою можливою затримкою є головною перевагою даного фреймворку[32].

Переваги використання Keras:

1. Дозволяє легко та швидко прототипувати (завдяки зручності користування, модульності та розширюваності);
2. Підтримує як згорткові мережі, так і рекурентні мережі, а також їх комбінації;
3. Є можливість паралельної роботи як на CPU так і на GPU.

Головна структура даних у Keras – це модель, тобто спосіб організації шарів. Найпростіший тип моделі це - послідовна модель, лінійне поєднання шарів. Для більш складних архітектур слід використовувати функціональний API Keras, який дозволяє будувати довільні архітектури нейронних мереж.

```
model = Sequential()
#створення моделі
model.add(Dense(units=64,activation='relu',
   input_dim=100))
model.add(Dense(units=10, activation='softmax'))
#додавання повнозв'язних шарів
model.compile(loss='categorical_crossentropy',
             optimizer='sgd',
             metrics=['accuracy'])
#завершення створення моделі
```

Якщо необхідно, то можна додатково налаштувати оптимізатор. Основний принцип Keras полягає в тому, щоб зробити речі досить простими, при цьому дозволяючи користувачеві повністю контролювати, коли їм потрібно.

```
model.compile(loss=  keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9,
nesterov=True))
#налаштування оптимізатора
model.fit(x_train, y_train, epochs=5, batch_size=32)
#навчання моделі
model.evaluate(x_test, y_test, batch_size=128)
#валідація моделі
```

3.3 Опис навчальної вибірки

У відкритому доступі існує багато різних за обсягами та за кількістю классів навчальних вибірок спецально пристосованих для задачі розпізнавання зображень.

З огляду на те, що навчання нейронних мереж буде відбуватися локально з використанням тільки CPU, це обмежує кількість доступних навчальних вибірок. Тому було вирішено зупинитися на MNIST[33].

База даних рукописних цифр MNIST має навчальний набір з 60000 прикладів і тестовий набір 10 000 прикладів. Це підмножина більшого набору, доступного від NIST. Цифри були нормалізовані за розміром і по центру зображень фіксованого розміру. Приклад MNIST

Рисунок 3.5 Приклад цифр з набору MNIST

База даних MNIST була побудована із NIST's Special Database 3 та Special Database 1, яка містить двійкові зображення рукописних цифр. Спочатку NIST позначав SD-3 як навчальний набір, а SD-1 - -3 набагато чистіше і легше розпізнати, ніж SD-1. Причину цього можна знайти у тому, що SD- -1 збирався серед учнів середньої школи. Здійснення розумних висновків із навчальних експериментів вимагає, щоб результат не залежав від вибору навчального набору та тесту серед повного набору зразків. Тому необхідно було створити нову базу даних шляхом змішування наборів даних NIST.

Навчальний набір MNIST складається з 30 000 моделей з SD-3 та 30 000 моделей з SD-1. SD-1 містить 58 527 знакових зображень, написаних 500 різними письменниками. На відміну від SD-3, де блоки даних кожного записувача з'являлися послідовно, дані в SD-1 шифруються.

Для навчання доступно 4 набори:

1. train-images-idx3-ubyte.gz: training set images (9912422 bytes);
2. train-labels-idx1-ubyte.gz: training set labels (28881 bytes);
3. t10k-images-idx3-ubyte.gz: test set images (1648877 bytes);
4. t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes);

Вибір MNIST як навчальної вибірки зумовлений в більшій мірі обмеженням у обчислювальній потужності. Якщо є можливість використання відеокарти з технолоією CUDA, тоді можна обрати і інші навчальні набори, такі як CIFAR-10 та CIFAR-100. Набір даних CIFAR-10 складається з 60000 кольорових зображень 32x32 пікселі, усього 10 класів зображень, з 6000 зображеннями на клас. Є 50000 навчальних зображень та 10000 тестових зображень. В свою чергу CIFAR-100 подібний до CIFAR-10, за винятком того, що він містить 100 класів, що містять 600 зображень у кожному. Існує 500 навчальних зображень та 100 тестуючих зображень на кожен клас. 100 класів CIFAR- .

3.4 Створення та навчання нейронних мереж

3.4.1 Мережа для розпізнавання

Структура мережі наведена на рисунку 3.9. Разом із описами шарів мережі наведені розмірності їх виходів.

ШНМ була спроектована для вирішення задачі класифікації зображень. В якості функції активації для повнозв'язного шару використовується Swift зі значенням $\beta = 1.5$. В якості функції для агрегувального шару використовується змішана агрегація з $a = 0.85$ та $a = 0.75$. для першого та другого шарів відповідно.

Мережа навчалася за допомогою алгоритму зворотного поширення помилки на протязі 10 епох, розмір пакету даних - 128. Метод оптимизации – Adam. Функція втрат - sparse_categorical_crossentropy. Навчання мережі наведено нижче, на рисунку 3.6

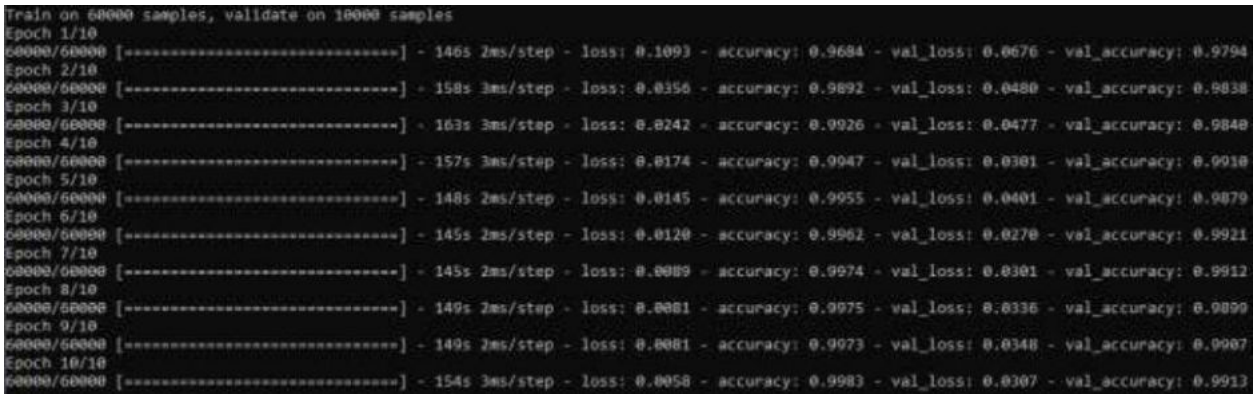


Рисунок 3.6 Процесс навчання мережі

Загальний час навчання склав 25 хвилин та 22 секунди. Результатом навчання нейронної мережі є класифікація рукописних цифр, тобто ймовірність належності до того чи іншого класу.

З огляду на точність розпізнавання мережею зображень (рисунок 3.7),

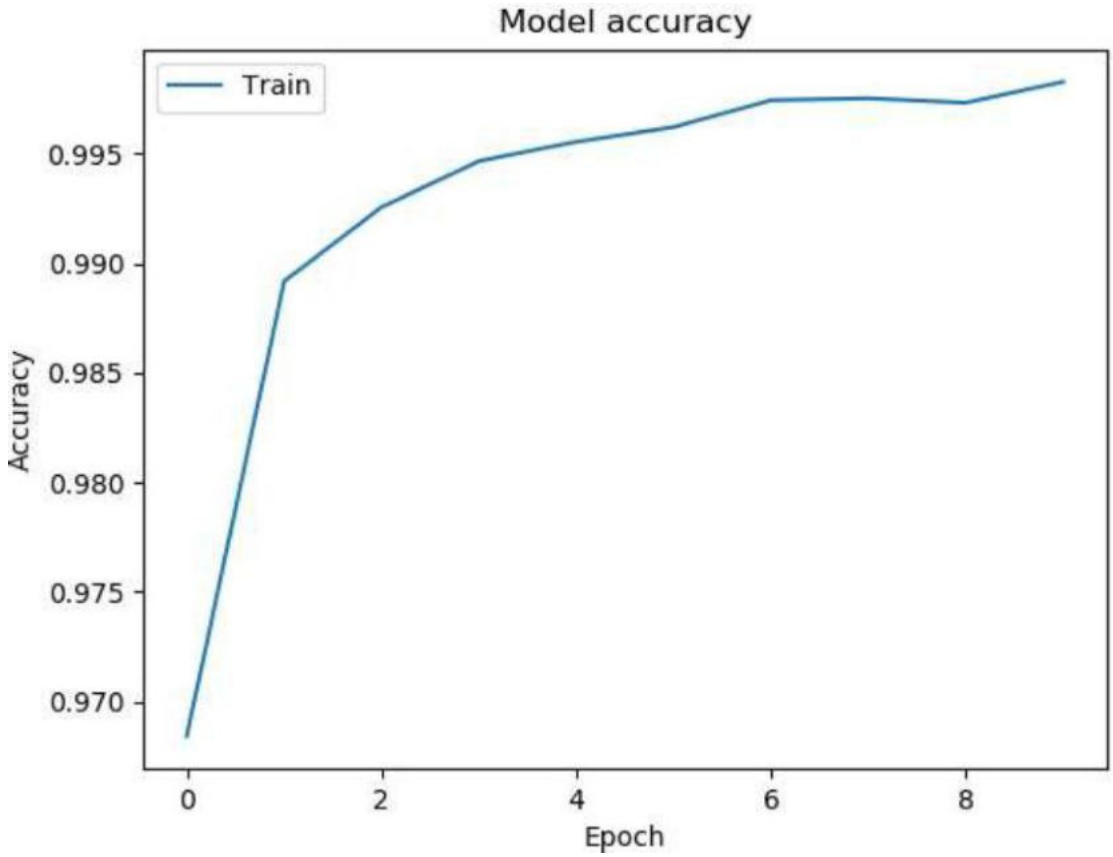


Рисунок 3.7 Точність мережі

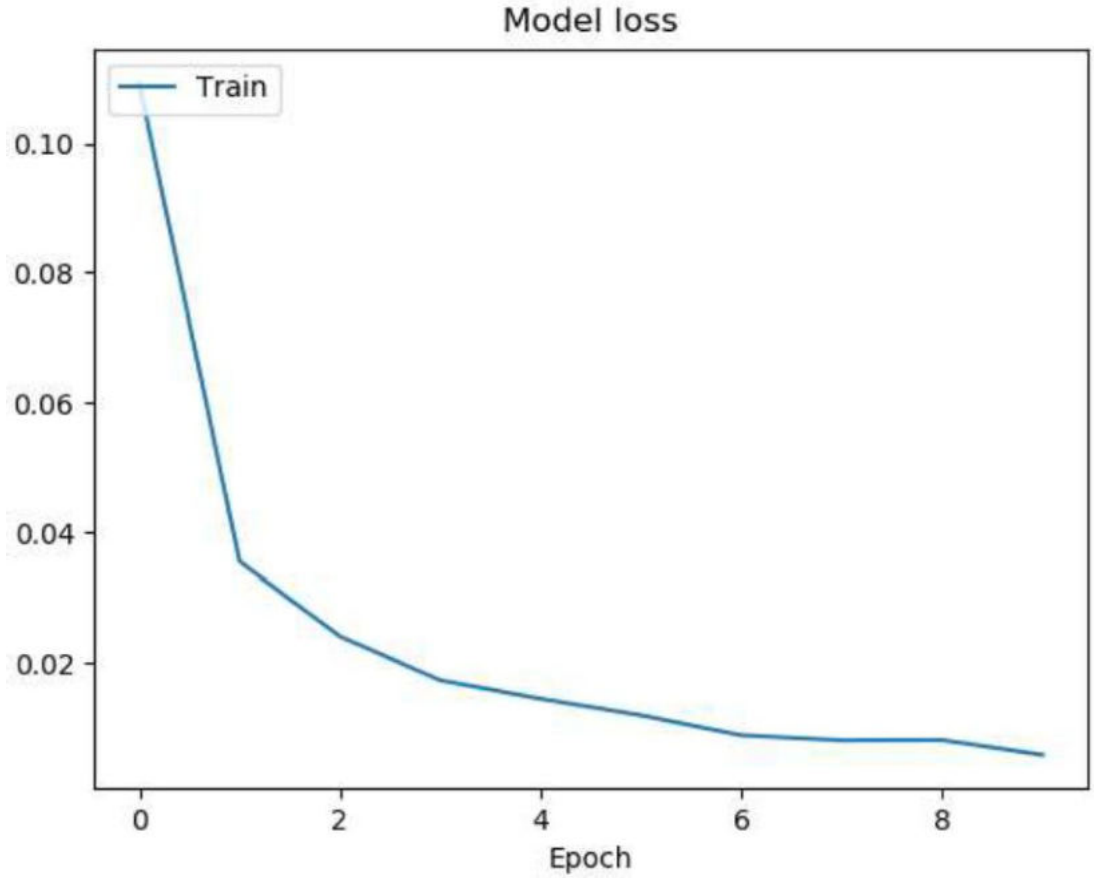


Рисунок 3.8 Значення функції втрат

Варто зазначити, що архітектура данної мережі є дещо спрощенною, порівняно з сучасними архітектурами згорткових нейронних мереж для розпізнавання зображень. Але все одно мережа досягає значних показників.

Було проведено навчання моделі із традиційним стохастичним градієнтним спуском та Адам у якості оптимізатора. В результаті чого Адам надав більш кращий результат навчання.

Рисунок 3.9 Структура мережі

3.4.2 Мережа для реставрації

Структура мережі наведена на рис. 3.10. Разом із описами шарів мережі наведені розмірності їх виходів.

ШНМ була спроектована для вирішення задачі відновлення спотворенних зображень. Побудова даної мережі засновано на використанні виключно TensorFlow без Keras та проекту Github Seratna/TensorFlow-Convolutional-AutoEncoder [34].

Мережа навчалася за допомогою алгоритму зворотного поширення помилки на протязі 4 епох, розмір пакету даних - 128.

Загальна кількість параметрів 1,369,421.

Кількість параметрів, що навчаються - 1,369,421.
Кількість параметрів, що не навчаються -0

Рисунок 3.10 Структура мережі

Важливо зазначити, що в данній ШНМ використовуються згорткові фільтри з розміром ядра згортки 5x5 пікселів. Це пояснюється тим, що для ефективної роботи мережі необхідно розглядати більш великі ознаки, ніж в ін

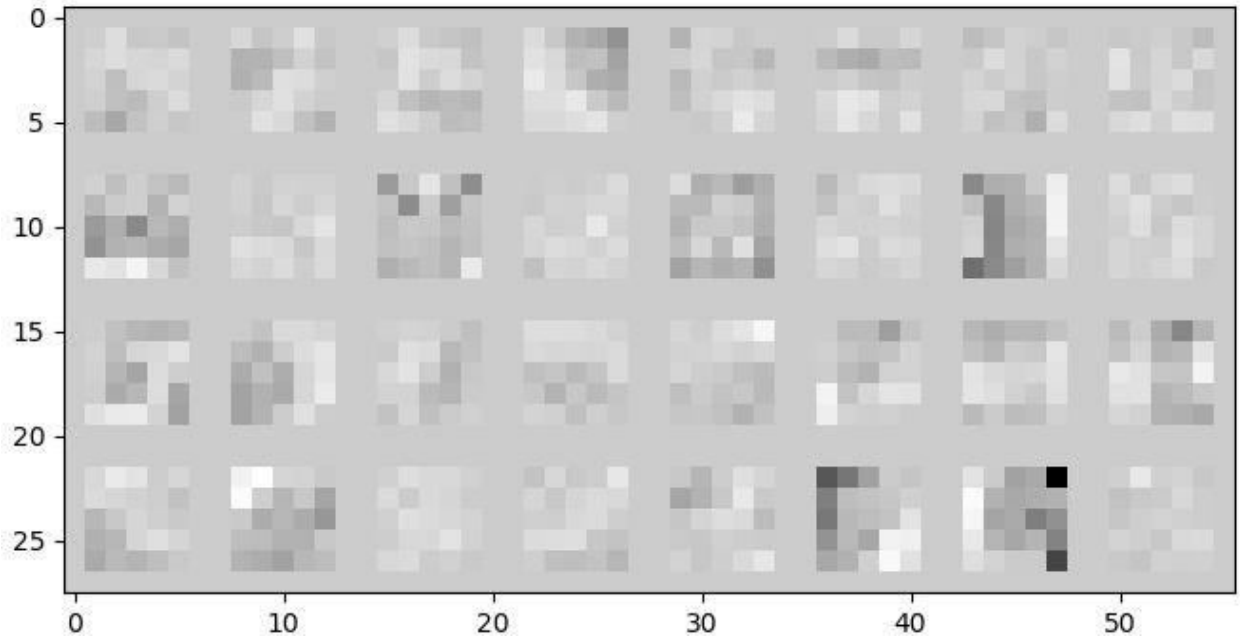


Рисунок 3.11 Візуалізація навчених фільтрів першого згорткового шару

3.4.3 мережа для знешумлення

Структура мережі наведена на рис. 3.15. Разом із описами шарів мережі наведені розмірності їх виходів.

ШНМ була спроектована для вирішення задачі знешумлення зображень. В якості функції активації для повнозв'язного шару використовується Swish зі значенням $\beta = 1.5$. В якості функції для агрегувального шару використовується змішана агрегація з $a = 0.8$. Розмір ядра згортки для усіх згорткових шарів становить 3x3 пікселі.

Мережа навчалася на протязі 4 епох, розмір пакету даних - 64.

Загальна кількість параметрів 895,174.

Кількість параметрів, що навчаються - 894,974.

Кількість параметрів, що не навчаються -200.

```
Epoch 1/4
60000/60000 [==============================] - 202s 3ms/step - loss: 0.0342 - accuracy: 0.8008
Epoch 2/4
60000/60000 [==============================] - 200s 3ms/step - loss: 0.0209 - accuracy: 0.8098
Epoch 3/4
60000/60000 [==============================] - 202s 3ms/step - loss: 0.0203 - accuracy: 0.8099
Epoch 4/4
60000/60000 [==============================] - 195s 3ms/step - loss: 0.0200 - accuracy: 0.8100
```

Рисунок 3.12 Процесс навчання мережі

Загальний час навчання склав .

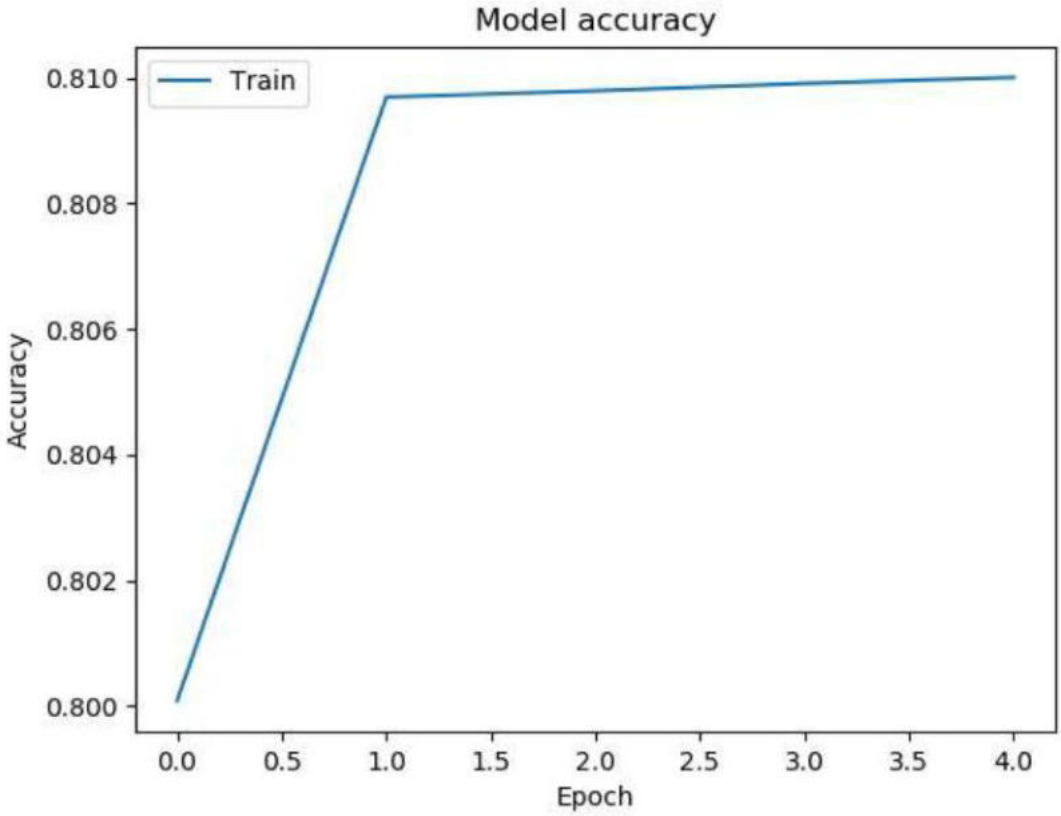


Рисунок 3.13 Точність мережі

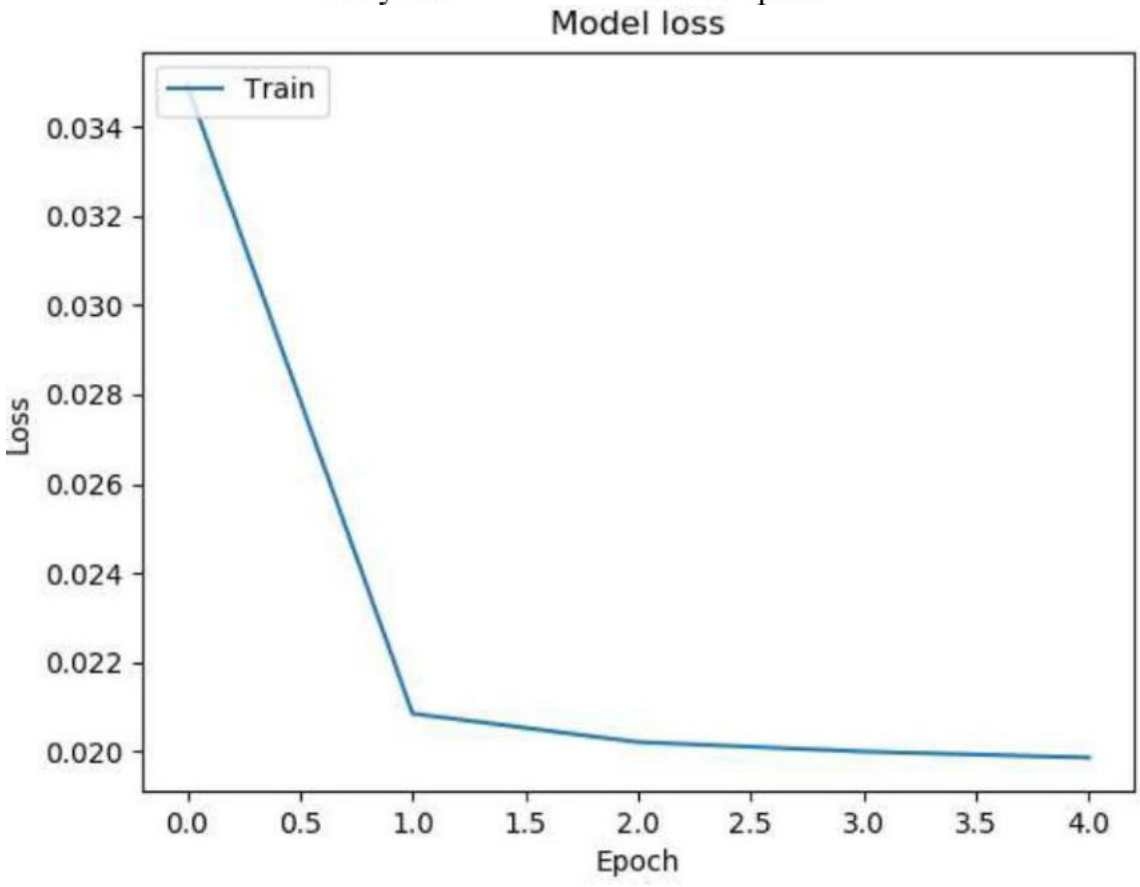


Рисунок 3.14 Значення функції втрат мережі

Можна побачити, що значення функції втрат та значення точності ніби завмерло на одному місці. Це пов’язано з тим,

абсолютно точно відновити вхідне зображення, оскільки характер спотворень заснований на випадковості й мережі складно побачити залежність між вхідним і вихідним шаром.

Але все одно, мережа надає задовільний результат в процесі прибирання шуму з вхідного зображення без видимих спотвернь самого зображення.

В даній структурі є тільки один агрегувальний шар (рисунок 3.15). Це пояснюється тим, що збільшення кількості агрегувальних шарів призведе до зменшення розмірності проміжного коду на виході кодувальника і на вході

зменшення розмірності для вилучення найбільш важливих ознак, в результаті це призведе до втрати інформації про оригінальне зображення, яке необхідно позбавити шуму. Тобто щоб мережа прибирала тільки шум, але без спотворення оригінальної форми зображення.

Рисунок 3.15 Схема згортки

Висновки за розділом

Отже, в даному розділі було обрано, фреймворки, мову програмування та середовище для розробки інтерфейсу додатку. Побудовано UML діаграму послідовності, що відображає взаємодію користувача з інтерфейсом програми. Створені структурні схеми трьох нейронних мереж, показано їх переваги.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

4.1 Інструкція користувача

Перше, що буде потрібно зробити користувачеві, це відкрити вхідне зображення натиснувши кнопку «Открыть изображение». Далі відкриється діалогове вікно яке дозволить відкрити зображення, яке необхідно обробити (рисунок 4.1).

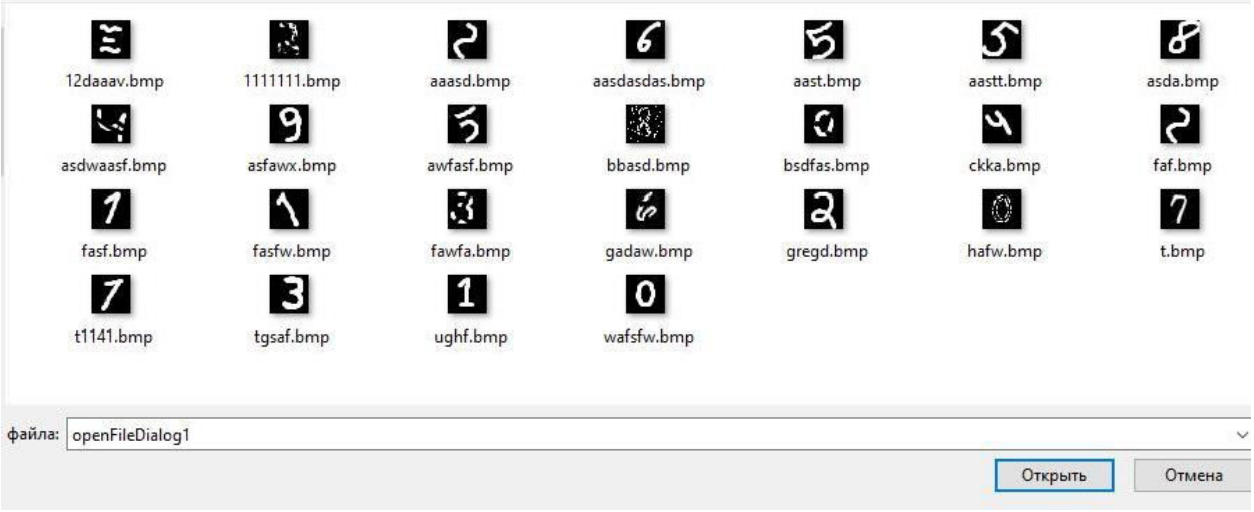


Рисунок 4.1 – Діалогове вікно

Якщо файл має допустимий формат, то зображення відкриється і поміститься у перше вікно для зображень (рисунок 4.2). Якщо відкритий файл має невірний формат, то нічого не відбудеться і необхідно знову спробувати відкрити зображення, але потрібного формату (png, bmp, jpeg).

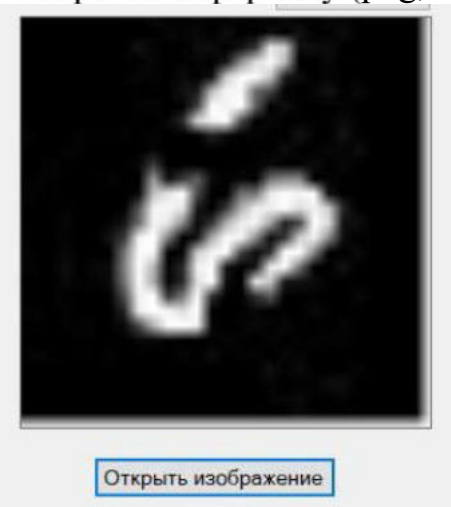


Рисунок 4.2 – Приклад шаблону для розпізнавання

Оскільки в даній роботі нейронні мережі навчені для обробки зображень на навчальному наборі MNIST, в якому кожне зображення має роздільну здатність 28x28 пікселів, то зображення автоматично розтягується по всій формі і дещо спотворюється завдяки операції маштабування. Але оскільки інтерфейс програми не прив'язаний до конкретних реалізацій нейронних мереж, це не є проблемою і можна в будь який момент відкрити зображення інших форматів та розмірів та обробити іншими мережами.

Наступним кроком є вибір режиму роботи програми (рисунок 4.3). За замовчуванням стоїть останній, а саме «Распознавание».

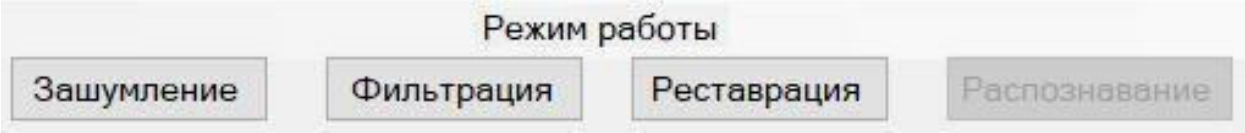


Рисунок 4.3 – Режим обробки зображення

Вибір режиму роботи програми дозволяє перемкнути прихований лічільник, який дозволяє керувати конкретним видом обробки.

Як видно із зображення вище, є 4 режими роботи, це:

- Зашумлення вхідного зображення;
- Фільтрація вхідного зображення від шуму;
- Реставрація пошкоджених частин вхідного зображення;
- Розпізнавання зображення.

Після того, як було обрано режим обробки зображення, необхідно натиснути клавішу «Применить». Далі програми за обраним режимом проведе обробку вхідного зображення та помістить результат у друге вікно (рисунок 4.4).

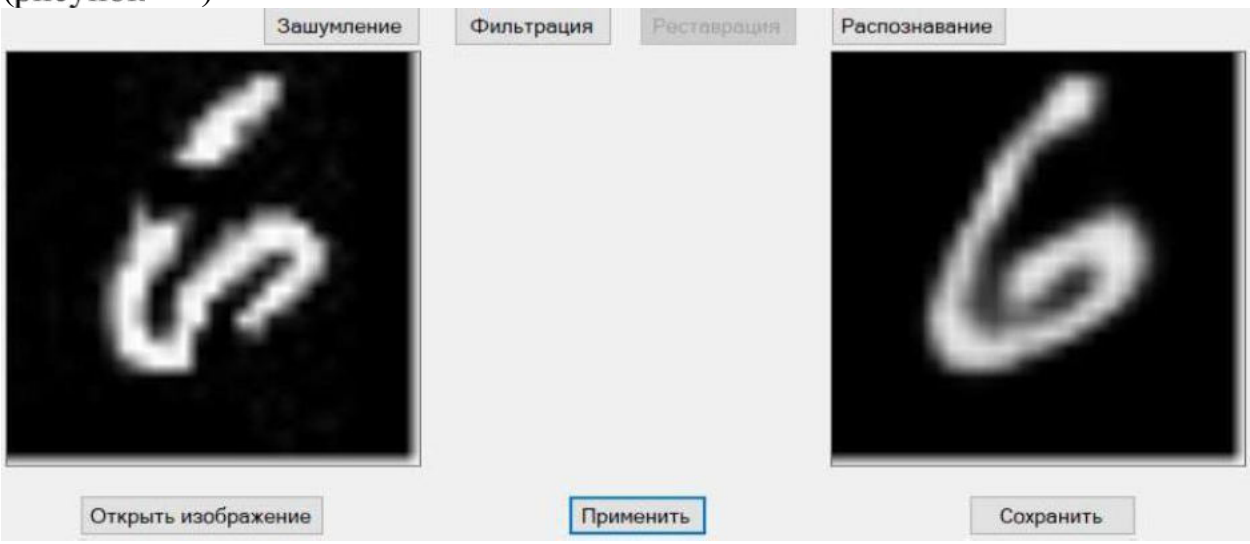


Рисунок 4.4 – Результат обробки зображення

Для того, щоб надіслати вихідне зображення з другого вікна до першого вікна, з я кого зображення подається на вхід до нейронним мереж, необхідно:

1. Натиснути лівою клавішею миши на друге вікно;
2. Прочитати зображення.

В результаті чого перше вікно буде містити зображення з другого вікна, а друге вікно буде очищено від зображення (рисунок 4.5).

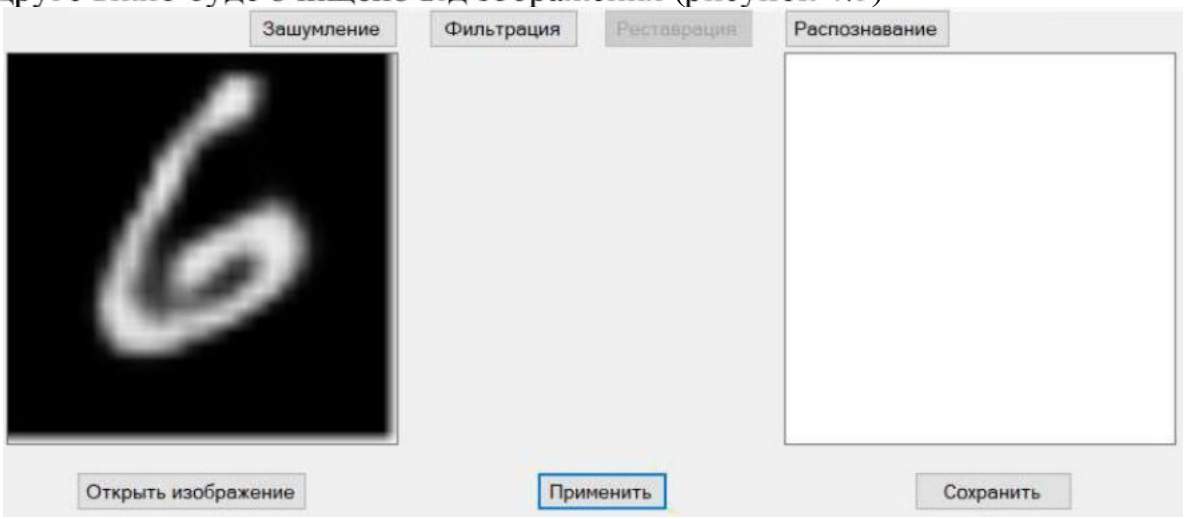


Рисунок 4.5 – Результат перенесення вихідного зображення після обробки ШНМ до вікна для вхідних зображень

4.2 Інструкція програміста

Для того, щоб запустити програму, необхідно встановити наступне програмне забезпечення та біблотеки:

- .NetFramework 4.7 та вище;
- Python 3.7.4 та вище;
  Keras 2.3.1;
  matplotlib 3.1.1;
  numpy 1.17.4;
  pip 19.0.3;
  scipy 1.3.2;
  tensorflow 1.15.0;
- Pillow 6.2.1.

Для справної роботи додатку, комп'ютер повинен відповідати наступним вимогам:

Таблиця 3.1-Системні вимоги

Операційна система	Windows 7,8,10
RAM	від 1 Гб
Вільне місце на жорсткому диску	не менше 100 Мб

При роботі додатку можуть виникнути ситуацій, що призведуть до небажаних наслідків або до некоректної роботи. Наприклад, якщо комп'ютер користувача не має відеокарти з пітримкою технології Nvidia cuDNN та має не дуже потужний центральний процесор, то час роботи нейронних мереж підвищиться у рази, вже не кажучи про час, який необхідно витратити на навчання нових мереж.

Також, при першому запуску операції обробки зображення, можлива невелика затримка, спричинена ініціалізацією необхідних бібліотек для застосування тут .

4.3 Тестування нейронних мереж

Тестування розпізнавання зображення (рисунки 4.6-4.10) при візуальних спотворенях та зашумленні. Для прикладу обрана образіна зліва цифра 5.

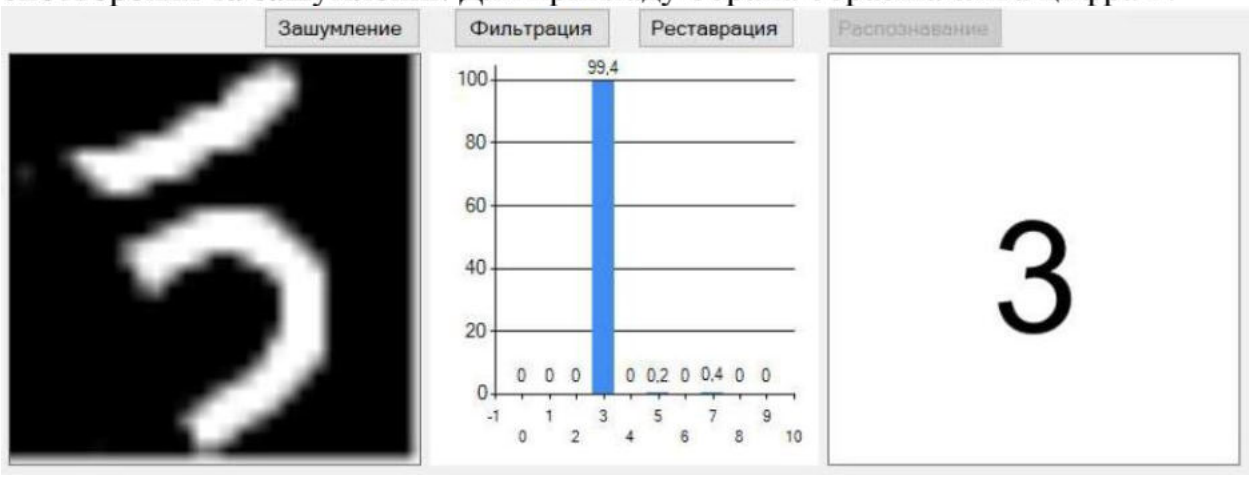


Рисунок 4.6 – Розпізнавання пошкодженої цифри

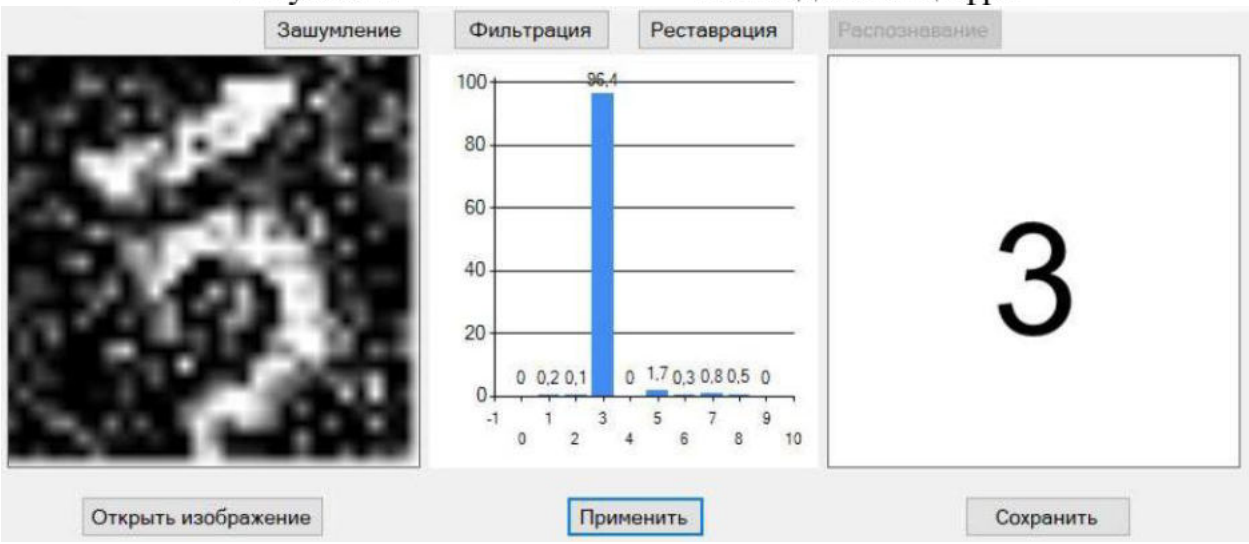


Рисунок 4.7 – Розпізнавання пошкодженої та зашумленної цифри

Рисунок 4.8 – Відновлення символу

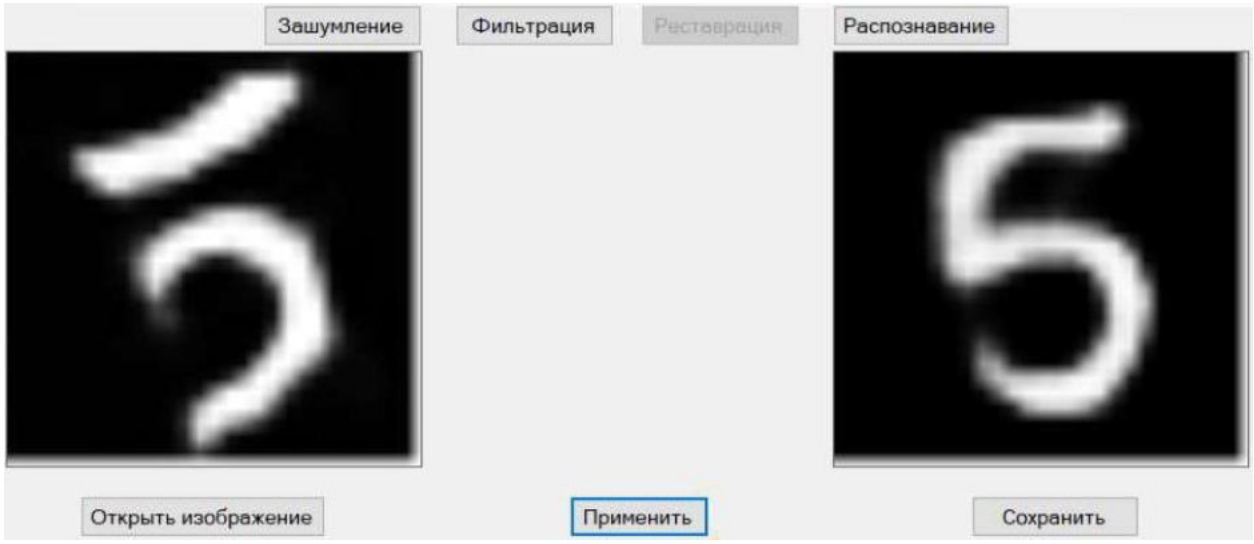


Рисунок 4.9 – Реставрація пошкодженої цифри

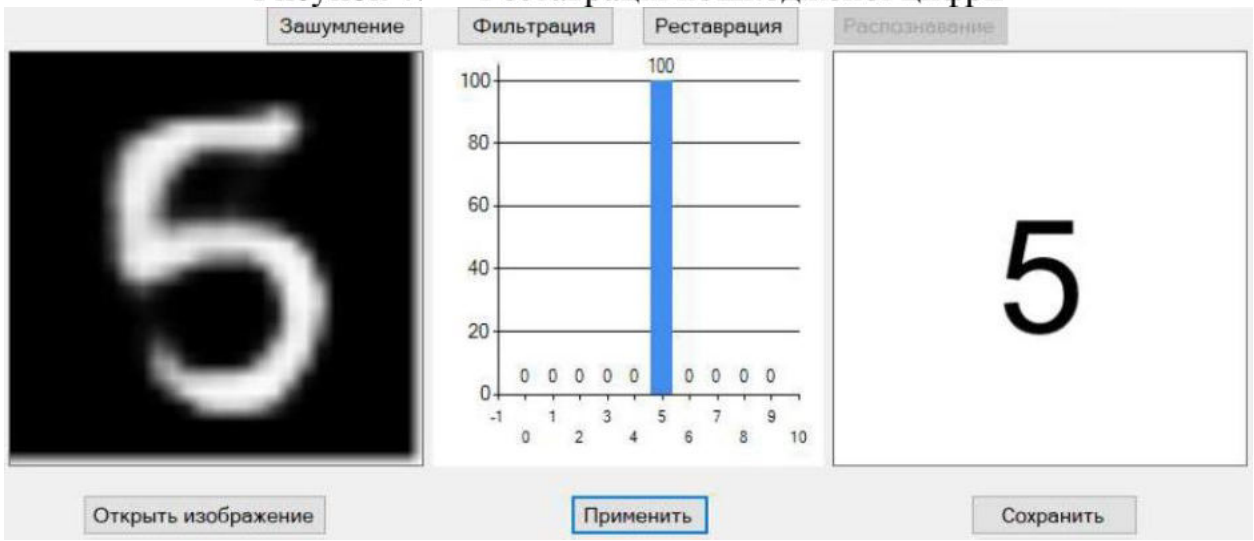


Рисунок 4.10 – Успішне розпізнавання зображення

Результати даного тестування можна вважати задовільним.

Тестування розпізнавання зображення при різних кутах нахилу вхідного зображення (рисунки 4.11-4.14):

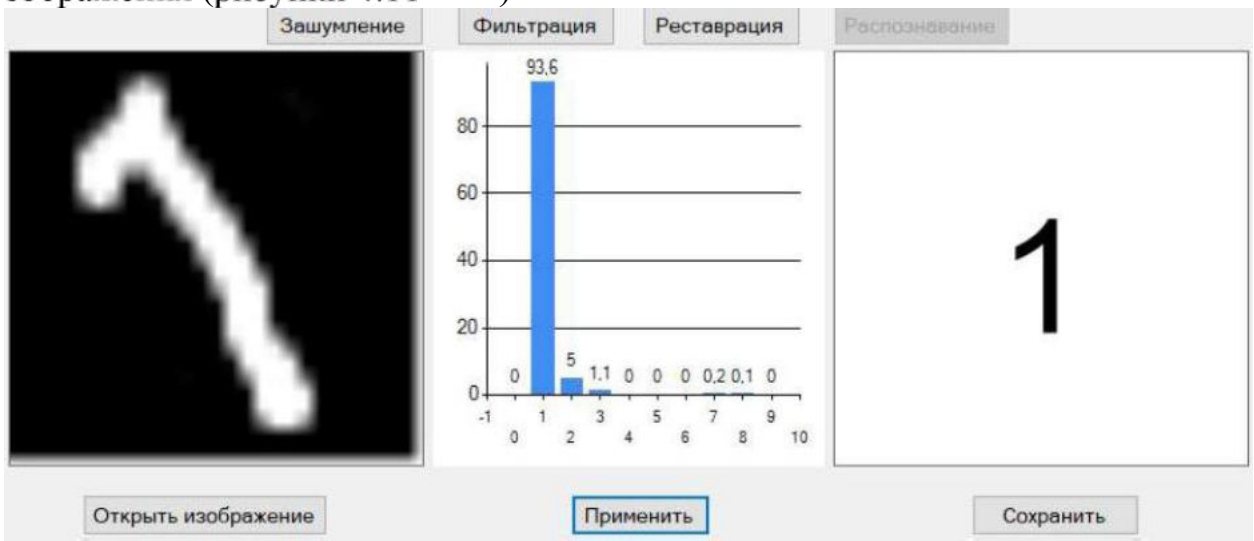


Рисунок 4.11 – Успішне розпізнавання зображення

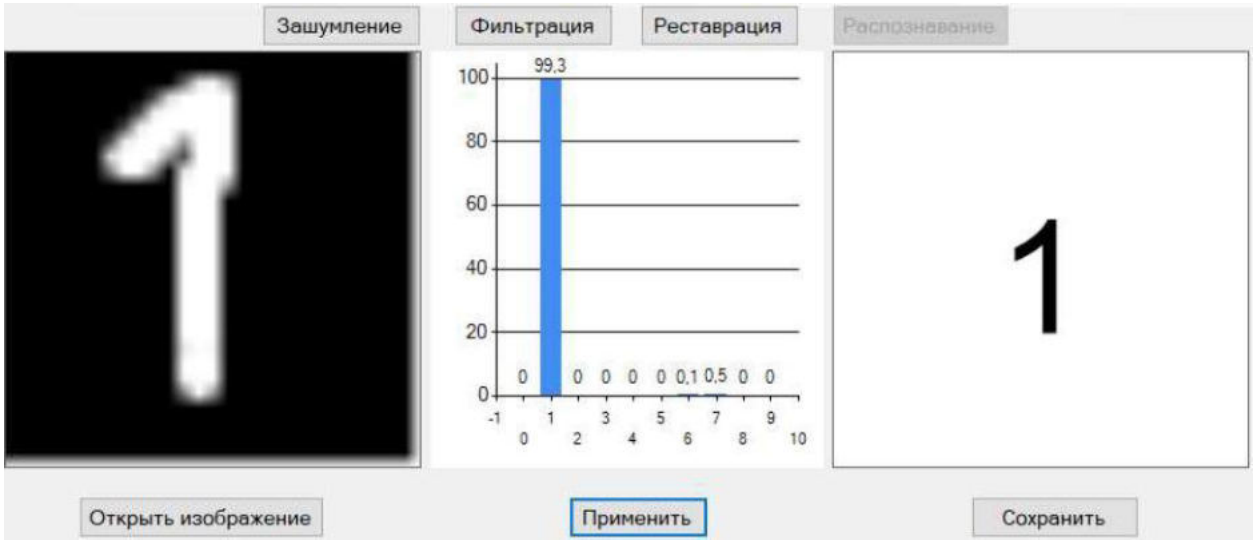


Рисунок 4.12 – Успішне розпізнавання зображення

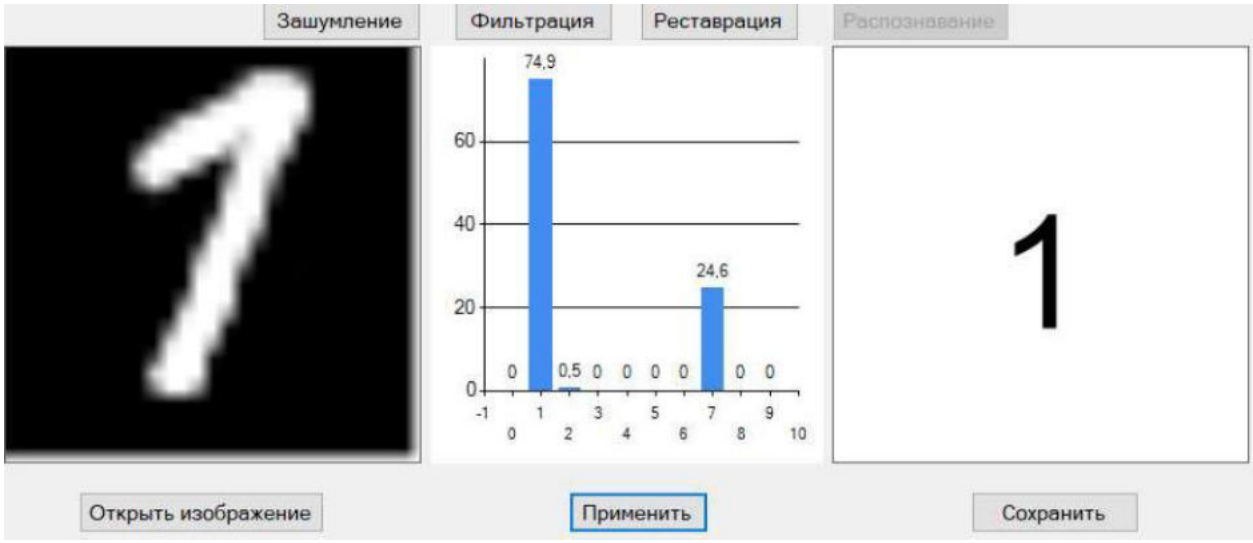


Рисунок 4.13 – Успішне розпізнавання зображення

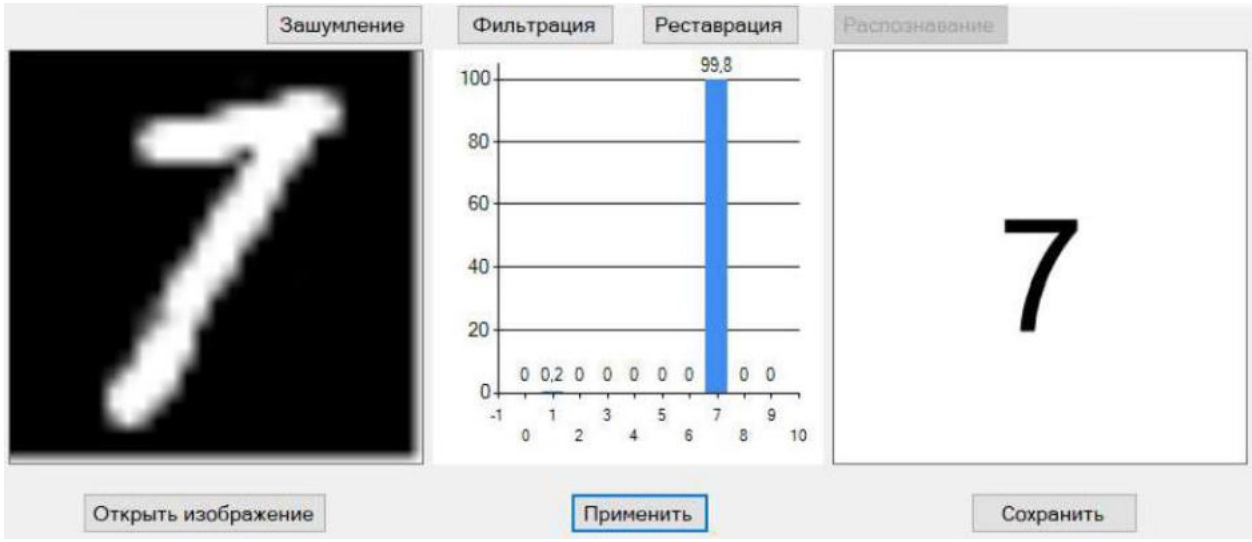


Рисунок 4.14 – Успішне розпізнавання зображення

Тестування розпізнавання зображень при багатократному зашумленні вхідного зображення (рисунок 4.15 -4.18):

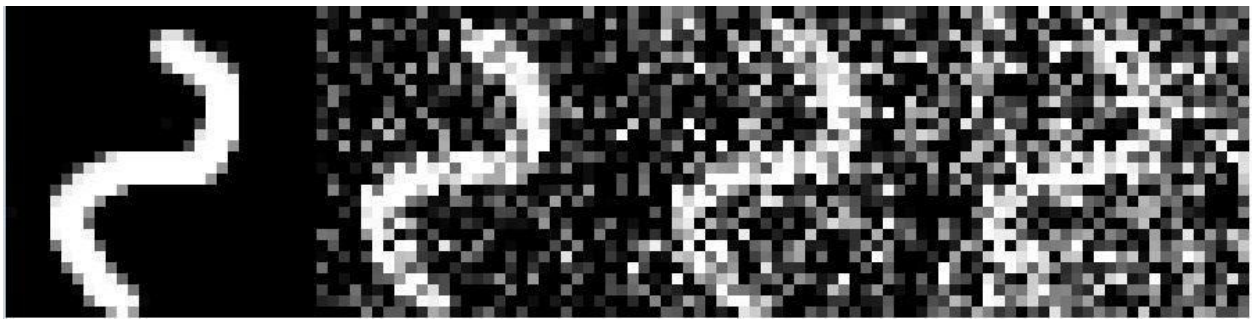


Рисунок 4.15 – Послідовне зашумлення вхідного зображення

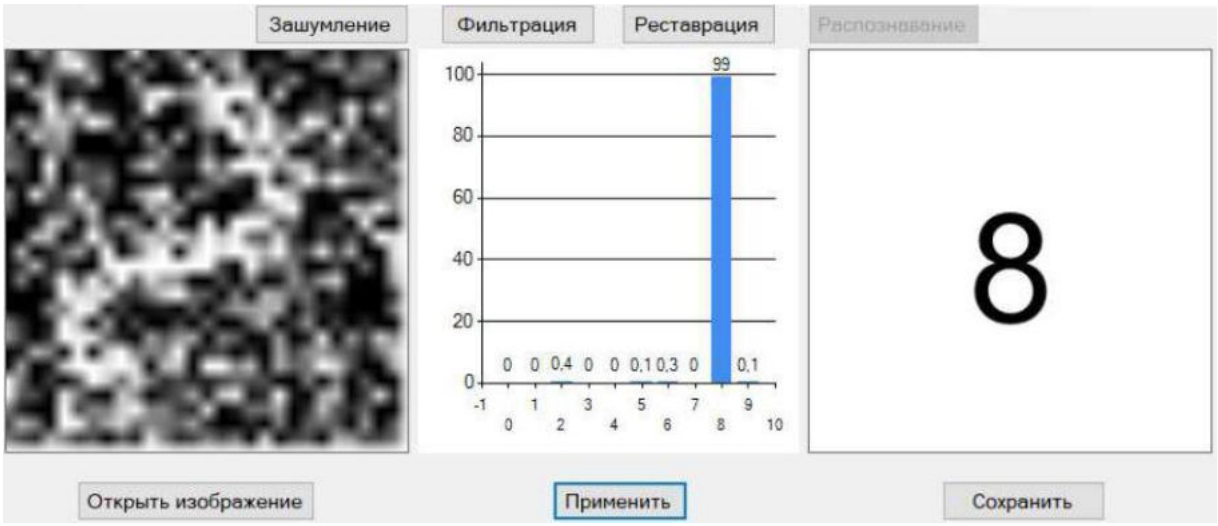


Рисунок 4.16 – Спроба розпізнати зашумлене зображення

Рисунок 4.17 – Послідовне знешумлення мережею

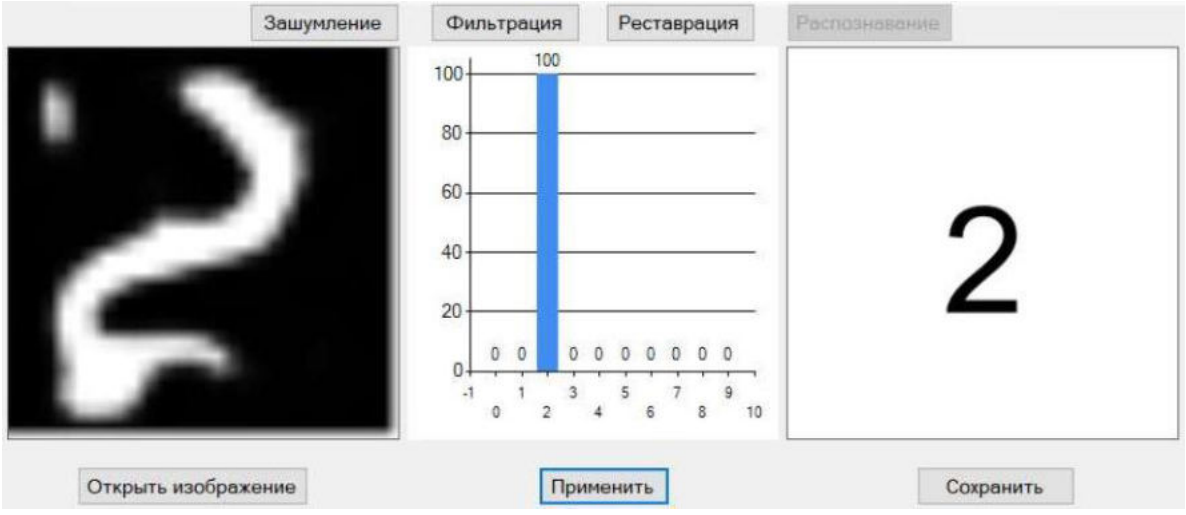


Рисунок 4.18 – Спроба розпізнати знешумлене

Висновки за розділом

Отже, в даному розділі було проведене тестування програми, а саме перевірено справне функціонування всіх функцій додатку та тестування нейронних мереж при різних вхідних данних. У результаті тестування якісна працездатність нейронних мереж була підтверджена.

Для спрощення роботи з програмою створені інструкції користувача і програміста, в яких описані всі основні можливості програми, системні вимоги, користування додатковим кодом .

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи була проведена модифікація згорткових нейронних мереж для задач обробки зображень, а саме фільтрації та розпізнавання зображень.

Проведено огляд нейронних та згорткових нейронних мереж. Розглянуті найкращі нейронні мережі для задач класифікації зображень. Проведено дослідження впливу використання модифікованої активаційної функції та модифікованого агрегувального шару. Результатом використання цих функції стало поліпшення якості роботи ШНМ.

Реалізація додатку відбувалася з використанням мови високого рівня програмування Python та бібліотек для глибокого навчання TensorFlow та Keras. Були спроектовані та навчені три нейронні мережі, побудована UML діаграми послідовності, що відображає взаємодію користувача з інтерфейсом.

Для спрощення роботи з програмою створені інструкції користувача і програміста, в яких описані всі основні функції програми, системні вимоги, інтерфейс. Проведено тестування нейронних мереж з різними вхідними даними та пошкодженнями. У результаті тестування якісна працездатність нейронних мереж була підтверджена.

В наш час використання штучних нейронних мереж неможливо обмежити якоюсь однією сферою, тому іноді важко визначити конкретні параметри для архітектури нейронної мережі. Реалізацій нейронних мереж для задач класифікації зображень безліч, особливо для набору рукописних цифр MNIST. Але це не виключає можливості дослідження з метою поліпшення показників ефективності роботи мережі.

Отже, з огляду на значну увагу світової спільноти до нейронних мереж та значну кількість праць, що постійно виходять, можна сказати, що використання модифікованої архітектури в даній роботі має свої перспективи для розвитку та подальшого застосування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. MachineVisionBook: Image Filtering Chapter 4. URL: http://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.files/MachineVision_Chapter4.pdf. (дата звернення: 14.11.2024).
2. Комп'ютерне моделювання систем та процесів. Розділ 8 : навч. посібник / Р. Н. Квєтний та ін. Вінниця : ЗНУ, 2012. 230 с.
3. The human brain in numbers: a linearly scaled-up primate brain. URL: https://www.frontiersin.org/articles/10.3389/neuro.09.031.2009/full. (дата звернення: 14.11.2024).
4. How many neurons make a human brain? Billions fewer than we thought. URL: https://www.theguardian.com/science/blog/2012/feb/28/how-many-neurons-human-brain. (дата звернення: 14.11.2024).
5. Sigmoid Function. URL: http://mathworld.wolfram.com/SigmoidFunction.html. (дата звернення: 14.11.2024).
6. Hyperbolic Tangent. URL: http://mathworld.wolfram.com/HyperbolicTangent.html. (дата звернення: 14.11.2024).
7. Glorot Xavier, Bordes Antoine, Bengio Yoshua. Deep Sparse Rectifier Neural Networks. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. 2011. С. 315-323.
8. A Practical Guide to ReLU. URL: https://medium.com/@danqing/a-practical-guide-to-relu- b83ca804f1f7. (дата звернення: 14.11.2024).
9. What is Deep Learning?. URL: https://machinelearningmastery.com/what-is-deep-learning/. (дата звернення: 14.11.2024).
10. Convolutional Neural Network. URL: http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork /. (дата звернення: 14.11.2024).
11. A Beginner's Guide To Understanding Convolutional Neural Networks. URL: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks /. (дата звернення: 14.11.2024).
12. McReynolds Tom, Blythe David. Advanced Graphics Programming Using OpenGL. Morgan Kaufmann, 2005. 672 с.
13. A Gentle Introduction to Padding and Stride for Convolutional Neural Networks. URL: https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks /. (дата звернення: 14.11.2024).
14. Dumoulin Vincent, Visin Francesco. A guide to convolution arithmetic for deep learning. Machine Learning (stat.ML). Université de Montréal AIRLab, pp. 140-148.

15. Multi-Layer Neural Network?. URL: http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks /. (дата звернення: 14.11.2024).
16. Optimization: Stochastic Gradient Descent. URL: http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/. (дата звернення: 14.11.2024). Text
17. Robbins, H. Monro, Stochastic Approximation Method. The Annals of Mathematical Statistics. 1951. 400 с.
18. Ian J. Goodfellow, Oriol Vinyals. Qualitatively characterizing neural network optimization problems. 2014. С. 20.
19. K. He, X. Zhang, S. Ren. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016. . С770-778.
20. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet classification with deep convolutional neural networks. Advances in neural information processing systems. 2012. С. 1097-1105.
21. Simonyan Karen, Zisserman Andrew. ICLR 2015: Very deep convolutional networks for large-scale image recognition, м. USA. 2015. С. 14.
22. C. Szegedy, W. Liu, Y. Jia. Going deeper with convolutions. Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. с9.
23. K. He, X. Zhang, S. Ren. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. с 770-778.
24. Diederik P. Kingma, Max Welling. An Introduction to Variational Autoencoders. Foundations and Trends in Machine Learning. 2019. № 4. Т. 12. С. 307-392.
25. Jonathan Masci, Ueli Meier, Dan Ciresan, J¨urgen Schmidhuber Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. 21st International Conference on Artificial Neural Networks : Artificial Neural Networks and Machine Learning ICANN 2011, м. Espoo, Finland, 14.06.2011 р. , 2011. С. 52-59.
26. Pascal Vincent, Hugo Larochelle. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. Journal of Machine Learning Research 11. 2010. С. 3371-3408.
27. IBM SPSS Statistics. URL: https://www.ibm.com/products/spss-statistics/details. (дата звернення: 14.11.2024).
28. Dingjun Yu, Hanli Wang, Peiqiu Chen, Zhihua Wei Mixed Pooling for Convolutional Neural Networks. 9th International Conference, RSKT 2014 : Rough Sets and Knowledge Technology, м. Shanghai, China, 24.10.2014 р., No.2,pp. 370 -375.

29. Andrew L. Maas, Awni Y. Hannun, Andrew Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. Computer Science Department, Stanford University, 2014. C. 6.
30. Prajit Ramachandran, Barret Zoph, Quoc V. Le. Searching for Activation Functions. Neural and Evolutionary Computing (cs.NE). 2017. C. 13.
31. Tensorflow. URL: https://www.tensorflow.org/overview/. (дата звернення: 14.11.2024).
32. Keras. URL: https://keras.io/. (дата звернення: 14.11.2024).
33. MNIST DATABASE. URL: http://yann.lecun.com/exdb/mnist/. (дата звернення: 14.11.2024).
34. TensorFlow Convolutional AutoEncoder. URL: https://github.com/Seratna/TensorFlow-Convolutional- (дата звернення 14.11.2024).

Додаток А

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import matplotlib
#tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging
.ERROR)
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout,
Flatten, MaxPooling2D, Activation, BatchNormalization,
Lambda
from keras.models import load_model
from PIL import Image
from keras.utils import plot_model
from mix import *
from keras.backend import sigmoid
from keras.utils.generic_utils import get_custom_objects
class Swish(Activation):

    def __init__(self, activation, **kwargs):
        super(Swish, self).__init__(activation,
**kwargs)
        self.__name__ = 'swish'
def swish(x):
    return (x * sigmoid(1.5*x))
get_custom_objects().update({'swish': Swish(swish)})
'''
class LossHistory(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.losses = []

    def on_batch_end(self, batch, logs={}):
        self.losses.append(logs.get('loss'))
'''
 (x_train, y_train), (x_test, y_test) =
tf.compat.v1.keras.datasets.mnist.load_data()
#image_index = 7777 # You may select anything up to
60,000
#print(y_train[image_index]) # The label is 8
#print(str(x_train.shape))
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
```

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print('Number of images in x_train', x_train.shape[0])
print('Number of images in x_test', x_test.shape[0])
model = Sequential()
model.add(Conv2D(filters = 32, kernel_size =
(3,3),padding = 'Same',
                 activation ='relu', input_shape =
(28,28,1)))
model.add(Conv2D(filters = 32, kernel_size =
(3,3),padding = 'Same',
                 activation ='relu'))
model.add(Lambda(custom_pool2d,
arguments={'alpha':0.85}))
model.add(Conv2D(filters = 64, kernel_size =
(3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size =
(3,3),padding = 'Same',
                # activation ='relu'))
model.add(Lambda(custom_pool2d,
arguments={'alpha':0.95}))
model.add(Flatten()) # Flattening the 2D arrays for
fully connected layers

model.add(BatchNormalization())
model.add(Dropout(0.15))
model.add(Dense(10,activation='softmax'))
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
history = model.fit(x=x_train,y=y_train, batch_size=128,
epochs=7, validation_data=(x_test, y_test), verbose=1)
history_dict = history.history
print(history_dict.keys())

model.save('Recognition_model.h5')
model.evaluate(x_test, y_test)
```

```
plt.plot(history.history['accuracy'])
#plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
#plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper left')
plt.show()
model = load_model('Recognition_model.h5')
#image_index = 6001
#img_rows = 28
#img_cols = 28
#plt.imshow(x_test[image_index].reshape(28,
28),cmap='Greys')
#plt.show()
#pred = model.predict(x_test[image_index].reshape(1,
img_rows, img_cols, 1))
#print(str((x_test[image_index].reshape(1, img_rows,
img_cols, 1)).shape))
#print('Eto cifra'+str(pred.argmax()))
#print('tut1')
#загрузить картинку для распознавания
im = Image.open("Input.bmp")
#конвертировать в массив (28,28,3)
np_im = np.array(im)
#print (str(np_im.shape))
#добавить 1 измерение (1,28,28,3)
np_im = np.expand_dims(np_im,axis=0)
#print (str(np_im.shape))
#перевести (0..255 в 0..1)
data_img = (np_im/255).astype(np.float64)
data1 = data_img
data2 = np.ones((1,28,28,1))
data2 = data2.astype(np.float64)
for a1, a2 in zip(data_img, data2):
```

```
        for c1, c2 in zip(b1, b2):
            c2[0] =  (c1[0] + c1[1] + c1[2])/3
x = data2
#print (str(x.shape))
pred = model.predict(x)
#print('Eto cifra'+str(pred.argmax()))
#print(model.get_layer(4).output)
#np.savetxt('ConvRecogniData/Output.txt', pred,
fmt='%.2f', delimiter='\n')
np.savetxt('Output.txt', np.append(pred.argmax(),
pred*100), fmt='%.2f', delimiter='\n')
#model.save('Recognition_model.h5')
#del model
#model = load_model('Recognition_model.h5')
#model.summary()

def custom_pool2d(x, alpha):
    pool_max = K.pool2d(x,pool_size=(2, 2), strides=(2,
2), pool_mode='max')
    pool_avg = K.pool2d(x,pool_size=(2, 2), strides=(2,
2), pool_mode='avg')
    return alpha * pool_max + (1 - alpha) * pool_avg

import numpy as np
import sys
#np.set_printoptions(threshold=sys.maxsize)
from keras.models import Sequential
from keras.layers import Conv2D, UpSampling2D,
BatchNormalization,MaxPooling2D, Lambda
from keras.models import load_model
from keras.datasets import cifar10
import tensorflow as tf
import matplotlib.pyplot as plt
from PIL import Image
from mix import custom_pool2d, random_pool2d
(x_train, _), (x_test, _) =
tf.compat.v1.keras.datasets.mnist.load_data()
x_train = x_train/255
x_test = x_test/255
print(str(x_train.shape))
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
print(str(x_train.shape))
x_test = x_test.reshape
```

```
model = Sequential()

model.add(Conv2D(32, kernel_size=3, strides=1,
padding='same', activation='relu', input_shape=(28, 28,
1)))
model.add(BatchNormalization())
padding='same', activation='relu'))
model.add(Lambda(custom_pool2d, arguments =
{'alpha':0.9}))
model.add(Conv2D(32, kernel_size=3, strides=1,
padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(UpSampling2D())
padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(1,  kernel_size=1, strides=1,
padding='same', activation='sigmoid'
model.compile(optimizer='adam', metrics=['accuracy'],
loss='mean_squared_error')
model.summary()

# We want to add different noise vectors for each epoch
#num_epochs = 2
NOISE = 0.35     # Set to 0 for a regular (non-
denoising...) autoencoder
for i in range(num_epochs):
    noise = np.random.normal(0, NOISE, x_train.shape)
    result = x_train + noise
    for a1 in result:
        for b1 in a1:
            for c1 in b1:
                if c1[0] < 0:
                    c1[0] = 0
                elif c1[0] > 1:
                    c1[0] = 1
    history=model.fit(result, x_train, epochs=1,
batch_size=128)
NOISE = 0.35
noise = np.random.normal(0, NOISE, x_train.shape)
result = x_train + noise

for a1 in result:

            for c1 in b1:
```

```
            if c1[0] < 0:
                c1[0] = 0
            elif c1[0] > 1:
                c1[0] = 1
history=model.fit(result, x_train, epochs=3,
batch_size=128)
noise = np.random.normal(0, NOISE, x_test.shape)
result = x_test + noise
for a1 in result:

            for [illegible]
                if c1[0] < 0:
                    c1[0] = 0
                elif c1[0] > 1:
                    c1[0] = 1
model.evaluate(result, x_test)
model.save('Antinoise_model.h5')
plt.plot(history.history['accuracy'])
#plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
#plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper left')
plt.show()
model = load_model('ConvDenoising/Antinoise_model.h5')

im = Image.open("ConvDenoising/Input.bmp")
np_im = np.array(im)
# np_im = 28.28.3


# 0.255 28.28.3 to 0.1 1.28.28.1
np_im = (np_im/255).astype(np.float32)
np_im = np.expand_dims(np_im,axis=0)
data1 = np_im
```

```
data2 = data2.astype(np.float64)
for a1, a2 in zip(data1, data2):
    for b1, b2 in zip(a1, a2):
        for c1, c2 in zip(b1, b2):
            c2[0] =  (c1[0] + c1[1] + c1[2])/3
x = data2
# x = 1.28.28.1
#x_test = x_test[:400]
x_test = x
##x_test = x
#pred_imgs = model.predict(middle)
pred_imgs = model.predict(x_test)
data = np.ones((1,28,28,3))
for a1, a2 in zip(pred_imgs, data):
    for b1, b2 in zip(a1, a2):
        for c1, c2 in zip(b1, b2):
            for index, d in enumerate(c2):
                c2[index] = c1[0]
data_img = (data.squeeze()*255).astype(np.uint8)
img = Image.fromarray(data_img, mode='RGB')#'RGB'
img.save('ConvDenoising/Output.bmp')


from models import *
from matplotlib import pyplot as plt
import tensorflow as tf
from PIL import Image
import numpy as np
import matplotlib
from mnist import MNIST

class Autoencoder(object):
    def __init__(self):

        x = tf.compat.v1.placeholder(tf.float64,
shape=[, 28, 28, 1])
        # кодувальник
        c1=Conv2D([5, 32], scope='conv1',
activation=tf.nn.relu)(x)
        p1=MaxPool(scope='pool1', padding='SAME',
strides=[1, 2, 2, 1])(c1)
        c2=Conv2D([5, 32], scope='conv_2',
```

```
        p2=MaxPool(scope='pool2', padding='SAME',
strides=[2, 2])(c2)
        ungr=UnGr(scope='unfold')(p2)
        enc=FullConn(20, scope='encode',
activation=tf.nn.relu)(ungr)
        # декодувальник
        dec=FullConn(7*7*32, scope='decode',
activation=tf.nn.relu)(enc)
        gr=Gr([-1, 7, 7, 32], scope='fold')(dec)
        unp1=UnPool((2, 2), scope='unpool_1',
output_shape=tf.shape(conv2))(gr)
        dec1=DeConv2D([5, 5, 32, 32],
activation=tf.nn.relu, scope='deconv_1', output
=tf.shape(p1))(unp1)
        unp2=UnPool((2, 2), scope='unpool_2',
output_shape=tf.shape(c1))(dec1)
        recon = DeConv2D([5, 5, 1, 32],
scope='deconv_2', activation=tf.nn.sigmoid,
output_shape=tf.shape(x))(unp2)

        # функція втрат
        los =tf.nn.l2_loss(x-recon)

        # тренування
        train = tf.compat.v1.train.AdamOptimizer(1e-
5).minimize(loss)

        # визначення
        self.x = x
        self.predict = recon
        self.fit = ttrain
        self.loss = loss

    def fit(self, passes, batch_size, train=False):

        mnist = MNIST()

        with tf.Session() as ses:
            if train:
                save, h = Model.new(ses)
            else:
                save, h = Model.prev(ses,
```

```
            for step in range(1+h, 1+passes+h):
                x, y = mnist.get_batch(batch_size)
                self.train.run(feed_dict={self.x: x})

                if step % 20 == 0:
                    loss =
self.loss.eval(feed_dict={self.x: x})

                if step % 250 == 0:  # збереження ваг
                    save.save(ses,
'ConvRestoration/saver/cnn', h=step)
                    print('Збереження успішне')

        with tf.compat.v1.Session() as sess:
            save, h = Model.prev(sess,
ckpt_file='ConvRestoration/saver/checkpoint')

            print('tut1')
            #загрузить картинку для реставрации
            im = Image.open("ConvRestoration/Input.bmp")
            #конвертировать в массив (28,28,3)
            np_im = np.array(im)
            #print (str(np_im.shape))
            #добавить 1 измерение (1,28,28,3)
            np_im = np.expand_dims(np_im,axis=0)
            print (str(np_im.shape))
            #перевести (0..255 в 0..1)
            data_img = (np_im/255).astype(np.float64)
            #print (str(data_img.shape))

            data1 = data_img
                  = np.ones((1,28,28,1))
            data2
            data2 = data2.astype(np.float64)
            #print('tut2')
            #print("size: %s, type: %s"%(data1.shape,
data1.dtype))
            #print("size: %s, type: %s"%(data2.shape,
data2.dtype))
            for a1, a2 in zip(data_img, data2):
                for b1, b2 in zip(a1, a2):
                    for c1, c2 in zip(b1, b2):
                        c2[0] =  (c1[0] + c1[1] +
c1[2])/3
```

```
            #print("size: %s, type: %s"%(x.shape,
x.dtype))
            #print (str(np_im.shape))
            #print (str(data_img.shape))


            #print('tut3')
            original, reconstructed = sess.run((self.x,
feed_dict={self.x: x}, self.predict))
            #print(str(recon.shape))
            data = np.ones((1,28,28,3))
            # (1,28,28,1) to (1,28,28,3)
            for a1, a2 in zip(reconstructed, data):
                for b1, b2 in zip(a1, a2):
                    for c1, c2 in zip(b1, b2):
                        for index, d in enumerate(c2):
                            c2[index] = c1[0]

            #print(str(data.shape))
            #for i in range(28):
                #data[0,i,i,1] = 0.0

            #print("size: %s, type: %s"%(data.shape,
data.dtype))
            # size: (1, 16, 16, 3), type: float64
            #
            data_img =
(data.squeeze()*255).astype(np.uint8)

            print("size: %s, type: %s"%(data_img.shape,
data_img.dtype))
            # size: (16, 16, 3), type: uint8
            # сохранить отреставрированное изображение
            img = Image.fromarray(data_img,
mode='RGB')#'RGB'
            img.save('ConvRestoration/Output.bmp')


def main():
    autoencoder = Autoencoder()
    #autoencoder.train(batch_size=128, passes=1500,
new_training=False)
```

```
class Model(object):
    def __init__(self):
        pass

    @staticmethod
    def new(ses):
        saver = tf.train.Save()  # create a saver
        h = 0

        ses.run(tf.global_variables_initializer())
        print('started a new session')

        return save, h

    @staticmethod
    def prew(ses, ckpt_file):
        save = tf.compat.v1.train.Saver()  # create a
save

        with open(ckpt_file) as file:  # read checkpoint
file
            line = file.readline()  # read the first
line, which contains the file name of the latest
checkpoint
            ckpt = line.split('"')[1]
              = int(ckpt.split('-')[1])
            h
        # restore
        save.restore(ses, 'ConvRestoration/saver/'+ckpt)
        print('restored from checkpoint ' + ckpt)
        return save, h
```

Додаток Б

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;
using System.Drawing.Drawing2D;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Windows.Forms.DataVisualization.Charting;
namespace NN
{
    public partial class MainForm : Form
    {
        Bitmap image;
        Bitmap diagram;
        bool drow = true;
        int metodMode = 4;
        bool recognizeNow = false;
        public MainForm()
    {
        InitializeComponent();

    }
    private void Form1_Load(object sender, EventArgs e)
    {
        B.Clear(pictureBox1);
        B.Clear (pictureBox2);
    }
    private void closeToolStripMenuItem_Click(object
sender, EventArgs e)
    {
        Close();
    }
```

```
    private void clearToolStripMenuItem_Click(object
sender, EventArgs e)
    {
        image = null;
         B.Clear (pictureBox1);
        B.Clear (pictureBox2);
        }
    private void drawFromComboBox_Click(object sender,
EventArgs e)
    {
        B.Clear (pictureBox1);
            B.Clear (pictureBox2);

        }
        private Color Brighter(Color clr)
        {
            clr = Color.FromArgb(clr.A ,
(int)((clr.R)+(255-clr.R)/2.9), (int)((clr.G) + (255 -
clr.G) / 2.9), (int)((clr.B) + (255 - clr.B) / 2.9));
            return clr;
        }
        public static int GetLuminance(Color one, Color
two)
        {
            int retval = one.R.CompareTo(two.B);

            // If the strings are not of equal length,
            // the longer string is greater.
            //
                return retval;


        }

               private void button2_Click_1(object
sender, EventArgs e)
        {
            Bitmap img = new Bitmap(pictureBox1.Image);
            if (metodMode == 1)
            {
                chart1.Visible = false;

img.Save("ConvDenoising  InputNoise.bmp",
```

```
                var program = new Process();
                program.StartInfo.CreateNoWindow = true;
                program.StartInfo.UseShellExecute =
false;
                program.StartInfo.RedirectStandardOutput
= false;
                program.StartInfo.FileName =
"python.exe";
                program.StartInfo.Arguments =
@"ConvDenoising\noising.py";
                //Process.Start("C:  Users  Dell
Vostro  AppData  Local  Programs  Python  Python37  pyth
on.exe", @"convolutional_autoencoder.py");
                program.Start();
                program.WaitForExit();
                Image temp;
                using (var bmpTemp = new
Bitmap("ConvDenoising  OutputNoise.bmp"))
                {
                    temp = new Bitmap(bmpTemp);
                }
                pictureBox2.SizeMode =
PictureBoxSizeMode.StretchImage;
                pictureBox2.Image = temp;

                //pictureBox1.Image = ResizeImg(image,
1024, 1024);
            }
            if (metodMode == 2)
            {
                chart1.Visible = false;
                img.Save("ConvDenoising\ Input.bmp",
System.Drawing.Imaging.ImageFormat.Bmp);
                var program = new Process();
                program.StartInfo.CreateNoWindow = true;
                program.StartInfo.UseShellExecute =
false;
                program.StartInfo.RedirectStandardOutput
= false;
                program.StartInfo.FileName =
"python.exe";
                program.StartInfo.Arguments =
@"ConvDenoising\
```

```
                //Process.Start("C:  Users  Dell
Vostro  AppData  Local  Programs  Python  Python37  pyth
on.exe", @"convolutional_autoencoder.py");
                program.Start();
                program.WaitForExit();

                Image temp;
                using (var bmpTemp = new
Bitmap("ConvDenoising  Output.bmp"))
                {
                    temp = new Bitmap(bmpTemp);
                }
                pictureBox2.SizeMode =
PictureBoxSizeMode.StretchImage;
                pictureBox2.Image = temp;

                //pictureBox1.Image = ResizeImg(image,
1024, 1024);
            }
            if (metodMode == 3)
            {
                chart1.Visible = false;
                img.Save("ConvRestoration
System.Drawing.Imaging.ImageFormat.Bmp);
                var program = new Process();
                program.StartInfo.CreateNoWindow = true;
                program.StartInfo.UseShellExecute =
false;
                program.StartInfo.RedirectStandardOutput
= false;
                program.StartInfo.FileName =
"python.exe";
                program.StartInfo.Arguments =
@"ConvRestoration\convolutional_autoencoder.py";
                //Process.Start("C:  Users  Dell
Vostro  AppData  Local  Programs  Python  Python37  pyth
on.exe", @"convolutional_autoencoder.py");
                program.Start();
                program.WaitForExit();
                Image temp;
                using (var bmpTemp = new
Bitmap("ConvRestoration  Output.bmp"))
                {
```

Input.bmp",

```
                }
                pictureBox2.SizeMode =
PictureBoxSizeMode.StretchImage;
                pictureBox2.Image = temp;

                //pictureBox1.Image = ResizeImg(image,
1024, 1024);
            }
            else if (metodMode == 4)
            {
                img.Save("ConvRecognition  Input.bmp",
System.Drawing.Imaging.ImageFormat.Bmp);
                var program = new Process();
                program.StartInfo.CreateNoWindow = true;
                program.StartInfo.UseShellExecute =
false;
                program.StartInfo.RedirectStandardOutput
= false;
                program.StartInfo.FileName =
"python.exe";
                program.StartInfo.Arguments =
@"ConvRecognition\rec_network.py";
                program.Start();
                program.WaitForExit();
                string answer = "";
                var fileName =
"ConvRecognition  Output.txt";
                List<double> outputs;
                using (StreamReader sr = new
StreamReader(fileName))
                {
                    answer =
Math.Round(double.Parse(sr.ReadLine(),
System.Globalization.CultureInfo.InvariantCulture),0).To
String();
                    outputs = new List<double>();

                    for (int i = 0; i < 10; i++)
                    {

outputs.Add(Math.Round(double.Parse(sr.ReadLine(),
System.Globalization.CultureInfo.InvariantCulture),1));
```

```
                pictureBox2.SizeMode =
PictureBoxSizeMode.Stretch;
                B.Clear(pictureBox2);
                pictureBox2.Image = B.DrowOutput (new
Bitmap(pictureBox2.Image),answer);
               recognizeNow = true;

                chart1.Series.Clear();
                Series series = new Series();
                series.ChartType =
SeriesChartType.Column;
                series.Name = "Outputs";
                chart1.Series.Add(series);
                chart1.ChartAreas[0].AxisY.Maximum =
outputs.Max()+5;
//chart1.ChartAreas[0].AxisX.IsMarksNextToAxis = false;
                chart1.ChartAreas[0].AxisY.Interval =
20;
                for (int i = 0; i < 10; i++)
                {

chart1.Series["Outputs"].Points.AddXY(i, outputs[i]);
                }

chart1.Series["Outputs"].IsValueShownAsLabel = true;

chart1.ChartAreas[0].AxisX.MajorGrid.Enabled = false;

                chart1.Visible = true;
                //original.Points.AddXY("CPU", 7.6);
                //modified.Points.AddXY("CPU", 1.6);
            }
            //pictureBox2.SizeMode =
PictureBoxSizeMode.StretchImage;
            //pictureBox2.Image = img;
        }



        private void button5_Click(object sender,
EventArgs e)
        {
            B.C
```

```
            B.C(pictureBox2);
        }

        private void pictureBox1_Click(object sender,
MouseEventArgs e)
        {

                if (e.Button == MouseButtons.Left)
                {
                    Clipboard.SetImage(new
Bitmap(pictureBox1.Image));
                }

                else if (e.Button == MouseButtons.Right)
                {
                if (Clipboard.ContainsImage())
                image = new
Bitmap(Clipboard.GetImage());
                pictureBox1.SizeMode =
PictureBoxSizeMode.StretchImage;
                pictureBox1.Image = image;
                B.C(pictureBox2);
                pictureBox1.Invalidate();



        public Image ResizeImg(Image b, int nWidth, int
nHeight)
        {
            Image result = new Bitmap(nWidth, nHeight);
            using (Graphics g =
Graphics.FromImage((Image)result))
            {
                g.InterpolationMode =
InterpolationMode.HighQualityBicubic;
                g.DrawImage(b, 0, 0, nWidth, nHeight);
                g.Dispose();
            }
            return result;
        }
        private void button4_Click(object sender,
EventArgs e)
```

```
            try
            {
                if (openFileDialog1.ShowDialog() ==
DialogResult.OK)
                {

                    //image = new
Bitmap(openFileDialog1.FileName);
                    using (var bmpTemp = new
Bitmap(openFileDialog1.FileName))
                    {
                        image = new Bitmap(bmpTemp);
                    }
                    pictureBox1.SizeMode =
PictureBoxSizeMode.StretchImage;
                    pictureBox1.Image = image;
                    //pictureBox1.Image =
ResizeImg(image, 1024, 1024);

NeiroGraphUtils.ClearImage(pictureBox2);
                    pictureBox1.Invalidate();
                    chart1.Visible = false;



            catch (Exception m)
            {

                N.ClImage(pictureBox1);
                N.ClImage(pictureBox2);
                drow = true;


        private void pictureBox2_Click(object sender,
MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left &&
pictureBox2.Image.Height!=0)
            {
                    Clipboard.SetImage(new
Bitmap(pictureBox2.Image));
```

```
        }
        private void button1_Click_1(object sender,
EventArgs e)
        {
            if (saveFileDialog1.ShowDialog() ==
DialogResult.Cancel)
                return;
            // получаем выбранный файл
            string filename = saveFileDialog1.FileName;
            // сохраняем текст в файл
            Bitmap img = new Bitmap(pictureBox2.Image);
            img.Save(saveFileDialog1.FileName,
System.Drawing.Imaging.ImageFormat.Png);
            MessageBox.Show("Файл сохранен");
        }
        private void button3_Click(object sender,
EventArgs e)
        {
            metodMode = 2;
            //label2.Text = "Фильтрация";
            button3.Enabled = false;
            button5.Enabled = true;
            button7.Enabled = true;
            button6.Enabled = true;
        }
        private void button5_Click_1(object sender,
EventArgs e)
        {
            metodMode = 4;
            //label2.Text = "Распознавание";
            button5.Enabled = false;
            button3.Enabled = true;

            button6.Enabled = true;
        }

        private void button6_Click(object sender,
EventArgs e)
        {
            metodMode = 3;
            //label2.Text = "Реставрация";
            button6.Enabled = false;
```

```
            button5.Enabled = true;
        }
        private void button7_Click(object sender,
EventArgs e)
        {
            metodMode = 1;
            //label2.Text = "Зашумление";
            button7.Enabled = false;
            button3.Enabled = true;
            button5.Enabled = true;
            button6.Enabled = true;

    }
}
```