МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра інформатики

РЕАЛІЗАЦІЯ ТА ПОРІВНЯННЯ МЕТОДІВ НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ

КУРСОВА РОБОТА

студентки IV курсу

Ніколаюк Дарії Андріївни

Науковий керівник:

к.ф.-м.н, ст. викладач Ющенко Ю.О.

КИЇВ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики,

доцент, к.ф.-м.н.

_____ С. С. Гороховський

(підпис)

“____” _____202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Ніколаюк Дарії

4 курсу факультету інформатики

ТЕМА: реалізація та порівняння методів навчання нейронних мереж

Індивідуальне завдання

Вступ, Аналіз предметної області

1.. Машинне навчання

2. Глибинне навчання

3. Згорткова нейронна мережа

4. Демонстрування трьох моделей навчання нейронних мереж

Висновки

Список використаних джерел

Дата видачі „__” _ 2021 р. Керівник: к.ф.-м.н, ст. викладач Ющенко Ю.О.

Завдання отримала _____

Тема: Реалізація та порівняння методів навчання нейронних мереж

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	05.10.2020	
2.	Огляд технічної літератури за темою роботи.	05.01.2021	
3.	Виконання аналізу актуальності теми	10.02.2021	
4.	Написання пояснювальної роботи.	15.03.2021	
5.	Попередня демонстрація проекту науковому керівнику	12.04.2021	
7.	Створення презентації	12.04.2021	
8.	Захист курсової роботи		

Студент _____Ніколаюк Д. А._____ _____

Керівник _____Ющенко Ю. О._____ _____

Зміст

Перелік умовних позначень

ML — Машинне навчання

AI — Штучний інтелект

ANN — Штучні нейронні мережі (Artificial Neural Networks)

CNN — Згорткові нейронні мережі (Convolutional Neural Networks)

DNN — Глибинні мережі переконань (Deep Belief Networks)

DNAF — Derivative of Neuron Activation Function

Вступ

Актуальність, наукове та практичне значення обраної теми

Останнім часом машинне навчання (ML) та штучний інтелект (AI) все частіше застосовуються в різних секторах людської діяльності для підвищення продуктивності та збільшення виробництва. Досягнення в галузі ML свідчили про зростання розвитку штучної нейронної мережі (ANN) та згорткових нейронних мереж (CNN). Ці техніки, що розробляють, можуть працювати ефективніше, ніж початкові моделі штучного інтелекту та машинного навчання. Також з кожним роком зростає зацікавленість вирішення складніших завдань розпізнавання об'єктів, обумовлена автоматизацією, необхідністю образних процесів комунікації в інтелектуальних системах. Тому вдосконалення реалізації розпізнавання комютерними системами образів є актуальною. Один із перспективних напрямків вирішення даної проблеми ґрунтується на застосуванні штучних нейронних мереж і нейрокомп'ютерів, як найбільш прогресивних по відношенню проблем класифікації завдань розпізнавання образів. У наш час запропоновано велику кількість архітектур нейромереж для застосування в розпізнаванні об'єктів. Аналіз існуючих рішень показує, що до сих пір не існує такої моделі, яка була б кращою серед усіх результуючих показників роботи.

Мета та завдання курсової роботи

Мета: Аналіз застосування нейронних мереж та порівняння різних методів і моделей їх навчання на задачі класифікації зображень.

Завдання: Опис та реалізація алгоритмів отримання із статичної фотографії фруктів текстове їх представлення за допомогою трьох моделей згорткових нейронних мереж. Аналіз існуючих алгоритмів, підходів та методів для реалізації цієї задачі.

Об'єкт дослідження

Об'єктом дослідження є моделі та засоби розпізнавання об'єкту на зображені за допомогою навчання нейронних мереж.

Джерела дослідження

Робота побудована на ретельному аналізі та дослідженні багатьох сторін машинного навчання та нейронних мереж. Було використано у навчальних цілях й опрацьовано багато наукових інтернет ресурсів, пов'язаних з темою роботи щодо штучного інтелекту, та для аналізу і порівняння наявних засобів у веб-просторі.

Аналіз сучасного стану питання та обґрунтування теми

Штучний інтелект - це теорія та розвиток комп'ютерів, здатних виконувати завдання, які може людина. Глибоке навчання являє собою елементарний рівень спроб досягнення цього завдання. Він використовується у зоровому сприйнятті, розпізнаванні мови, іграх, експертних системах, прийнятті рішень, медицині, авіації та перекладі між мовами. В ігровій індустрії штучний інтелект може бути корисним, оскільки ми можемо мати «ігрового робота» в якості суперника, коли людський гравець недоступний. Ми могли б також мати алгоритми глибокого навчання, які пропонують, як ворожі ікру можна стратегічно розмістити на арені, щоб отримати різні рівні складності. Військові, а також авіаційні галузі можуть використовувати штучний інтелект для сортування інформації, що стосується

повітряного руху, а потім забезпечити своїх пілотів найкращими методами, щоб уникнути дорожнього руху. Медична клініка може використовувати системи штучного інтелекту для організації графіків ліжок, чергування персоналу та надання медичної інформації [1]

РОЗДІЛ 1: Машинне навчання

Машинне навчання, за своїм визначенням, є областю інформатики, яка виникла завдяки вивченню розпізнавання образів та обчислювальної теорії навчання в галузі штучного інтелекту. Це навчання та побудова алгоритмів, які навчаються та роблять прогноз з наборів даних. Ці процедури працюють шляхом побудови моделі на прикладі вхідних даних для того, щоб робити прогнози, а не слідувати чітким статичним інструкціям програми. Тобто дякуючи машинному навчанню програміст не повинен писати інструкції, що вираховують всі можливі варіанти проблеми і мають усі можливі варіанти їх вирішення. "Про комп'ютерну програму кажуть, що вона вчиться на досвіді E щодо певного завдання T і деякого показника ефективності P, якщо її ефективність на T, виміряна P, покращується із досвідом E" - Том Мітчелл. Отже, якщо ми хочемо, щоб наша програма передбачала, наприклад, форми трафіку на зайнятому вузлі (завдання T), ми можемо запустити його через процес машинного навчання з даними про попередні шаблони трафіку (досвід E) і, якщо вона успішно “навчилася ”, Тоді він буде краще прогнозувати майбутні моделі руху (показник ефективності P).

Тож коли нам доречно використовувати машинне навчання, а не безпосереднє програмування комп'ютера для виконання поставленого завдання? Два аспекти даної проблеми можуть вимагати використання програм, які навчаються та вдосконалюються на основі їхнього "досвіду":

складність проблеми та потреба в адаптації.

Задачі занадто складні для програмування

Завдання, які виконують люди: Є безліч завданнь, які ми, люди, виконуємо навіть не задумуючись, проте наш самоаналіз щодо того, як ми це робимо, є недостатньо продуманим, щоб написати чітко визначену програму. Приклади таких завдань включають розуміння зображення, розпізнавання мови, та керування автомобілем. Для таких задач використовують найсучасніші програми машинного навчання, програми, які «вчаться на своєму досвіді» та досягають

цілком задовільних результатів, коли їх навчають на достатньо великій кількості прикладів.

Завдання, що перевищують людські можливості: Технології машинного навчання вирішують ще один широкий набір завдань, пов’язаний з аналізом дуже великих і складних наборів даних. Наприклад, аналіз геномних даних, астрономічні дані, електронна комерція, перетворення медичних архівів у медичні знання, розрахунок рейтингів веб-сторінок, прогнозування погоди. Черезвелику кількість доступних цифрових даних, які занадто великі і занадто складні, щоб люди могли їх зрозуміти, навчання виявляти закономірності в них - перспективна область, в якій є поєднання програм, які навчаються з майже необмеженим обсягом пам'яті та постійно зростаюча швидкість обробки комп'ютерів.

Адаптивність

Однією з обмежувальних особливостей запрограмованих інструментів є їх чіткість - як програма буде записана та встановлена, вона залишається незмінною. Однак багато завдань змінюються або з часом, або від одного користувача до іншого. Інструменти машинного навчання - програми, поведінка яких адаптується до вхідних даних, пропонують вирішення таких проблем. Вони за своєю природою адаптуються до змін середовища, з яким вони взаємодіють. Типові успішні програми машинного навчання для вирішення таких проблем включають програми виявлення спаму, що автоматично адаптуються до змін у характері спаму; програми, що декодують рукописний текст, де фіксована програма може адаптуватися до варіацій між рукописним введенням різних користувачів та програми розпізнавання мовлення. [1]

Розглянемо розпізнавання мови, коли акустичний мовний сигнал перетворюється на текст ASCII. Вимова слова може відрізнятися від людини до людини через різницю у віці, вимови або статі, тому для машинного навчання важливо зібрати велику колекцію зразків висловлювань різних людей і навчити складати їх у слова. В якості іншого прикладу можна розглянути маршрутизацію

пакетів через грід-обчислення. Шлях, що максимізує якість обслуговування від джерела до пункту призначення, регулярно змінюється у міру зміни системного трафіку. Процедура маршрутизації навчання може адаптуватися до найкращого шляху шляхом моніторингу мережевого трафіку. [2]

Типи навчання

Навчання - це, звичайно, дуже широка сфера. Тому область машинного навчання розгалужилася на декілька підгруп, що займаються різними типами навчальних завдань. Вирішувати завдання можна за допомогою різних методів. Усі методи діляться на три основних групи.

Навчання з вчителем (Supervised Learning)

У навчанні з вчителем, або як ще часто його називають контрольоване навчання, програма навчається на прикладах(мічені дані). Надано приклади бажаних входів та виходів, а алгоритм використовує цей вхід для визначення кореляцій та логіки, які можуть бути використані для прогнозування відповіді. Під час контрольованого навчання надаються зразки запитань та відповідей. Після того, як логічний шаблон визначений, його можна застосувати для вирішення подібних проблем. Наприклад, в алгоритмі машинного навчання, який визначає, чи є публікація спамом чи ні, навчальний набір включатиме публікації, позначені як "спам", та публікації, позначені як "не спам", щоб допомогти навчити алгоритм розпізнавати різницю. Контрольовані алгоритми навчання виводять функцію з позначених даних і використовують цю функцію на нових прикладах. Оскільки алгоритм отримує набір даних як вхідні дані, з якими вже пов’язана правильна відповідь, алгоритм дізнається, порівнюючи свої результати з правильною відповіддю, і якщо виявляє помилки, він відповідно коригує модель. Навчання триває до тих пір, поки алгоритм не виведе інформацію в межах бажаного діапазону точності. [3] Найбільш широко використовувані контрольовані методи навчання є дерева рішень, метод опорних векторів, регресії, нейронні мережі та баєсовська статистика

Навчання без вчителя (Unsupervised Learning)

При неконтрольованому навчанні машина сама вивчає дані для виявлення закономірностей. У цьому випадку немає ключа відповіді. Машина визначає кореляції та взаємозв'язки шляхом аналізу наявних даних. Навчання без вчителя моделюється на основі того, як ми, люди, природно спостерігаємо за світом: роблячи висновки та групуючи подібно до речей на основі необмеженого спостереження та інтуїції. У міру зростання нашого досвіду (або у випадку з машиною - зростає кількість даних, яким вона піддається), наша інтуїція та спостереження змінюються та / або стають більш досконалими. У реальності гарно розмічені дані це рідкість, саме тому для їх розмітки зазвичай використовують або спеціальні сервіси, у яких реальні люди з країн з дешевою робочою силою (зазвичай Індії або Китаю) за мінімальну плату вручну класифікують дані, або спеціальні алгоритми для розмітки. У випадках, коли розмітка даних неможлива, вдаються до методів навчання без учителя. До таких алгоритмів належать задачі кластеризації, самоорганізаційна карта Кохонена, зменшення розмірності і пошуку правил.

Навчання з підкріпленням (Reinforcement learning)

При навчанні підкріплення машині надається набір дозволених дій, правил та потенційних кінцевих станів. Іншими словами, визначені правила гри. Застосовуючи правила, досліджуючи різні дії та спостерігаючи реакції, що виникають, машина вчиться використовувати правила для створення бажаного результату. Таке навчання є окремим випадком контрольованого навчання, але вчителем в даному випадку є «середовище». Програма не має попередньої інформацією про середовище, але має можливість здійснювати в ній будь-які дії. Середовище реагує на ці дії і тим самим надає машині дані, які дозволяють їй реагувати на них і вчитися.

Навчання з підкріпленням використовується для вирішення більш складних завдань, ніж навчання з учителем і без. Воно використовується, наприклад, в трейдингових ботах, в системах навігації для роботів, в ботах для комп'ютерних

ігор, в логістиці, при складанні графіків і плануванні завдань. Q-навчання, марковський процес вирішування, штучні нейронні мережі є поширеними методами навчання з підкріпленням

Переваги машинного навчання

Розглянемо найістотніші переваги систем з машинним навчанням в порівнянні з загальноприйнятими альтернативами, такими як ручний аналіз, жорстко запрограмовані бізнес-правила і прості статистичні моделі.

Точність. Машинне навчання використовує дані для створення приймаючої рішення програми, оптимізовану під поставлену задачу. У міру накопичення даних автоматично зростає точність прогнозів.

Автоматизація. У міру підтвердження і відкидання відповідей ML-модель може автоматично виявляти нові шаблони. Це дозволяє впроваджувати машинне навчання безпосередньо в автоматизовані робочі процеси.

Швидкість. Машинне навчання дає відповіді за долі секунди після надходження нової інформації, дозволяючи системам реагувати в реальному часі. про

Можливість настройки. Багато задач, керовані даними, можна вирішити за допомогою машинного навчання. Моделі будуються на базі ваших власних даних і допускають настройку під будь-яку систему заходів, прийняту у вашому бізнесі.

Масштабованість. При зростанні бізнесу ML-модель легко пристосовується до збільшуваних об’ємів даних. Деякі алгоритми можна використовувати для обробки безлічі даних на різних обчислювальних машинах в хмарних середовищах.

РОЗДІЛ 2: Глибинне навчання

Сучасна, вдосконалена техніка машинного навчання, яка використовує надзвичайно складні нейронні мережі, називається глибинним навчанням, оскільки створені моделі значно складніші, ніж традиційні нейронні мережі. Моделі глибокого навчання також поглинають значно більший обсяг даних, ніж їх попередники. Чому це важливо? Глибинне навчання сьогодні є основою багатьох передових систем машинного навчання. Найголовніше чого глибоке навчання досягло це значно покращило здатність розуміти та аналізувати зображення, відео та звук. Це стало можливим завдяки значним досягненням у дослідженнях машинного навчання, а також значному зростанню як наявних даних, так і величезних обчислювальних потужностей.

Новий напрямок досліджень машинного навчання, який було запроваджено з метою наближення машинного навчання до однієї з його початкових цілей: штучного інтелекту. Глибинне навчання сягає корінням до Неокогнітрону — мережі штучних нейронів (ANN), представлена Фукусімою в 1980 р. Ідеєю ANN було розробити метод навчання шляхом моделювання людського мозку. Однак цей метод втратив підтримку спільноти через те, що для навчання мережі йому потрібна була величезна кількість даних та непрактичний час. Поглиблене навчання - це метод тренування ANN, використовуючи мало даних. Це причина, чому ANN знову в грі. Можна порівняти машинне навчання з глибинним на прикладі задачі виявлення обличчя. Якщо алгоритм машинного навчання вивчає такі частини обличчя, як очі та ніс, то алгоритм глибинного навчання вивчить додаткові параметри, такі як відстань між очима та довжина носа.

Найпопулярніші типи моделей глибинного навчання включають в себе згорткові нейронні мережі (CNN). Це тип штучної нейронної мережі, що широко використовується в комп'ютерному зорі, де окремі нейрони викладені таким чином, що вони реагують на ділянки, що перекриваються. Останнім часом CNN успішно застосовуються до автоматичного розпізнавання мовлення (ASR). Глибинні мережі переконань (DNN) та згорткові глибині мережі переконань —

це деякі інші популярні використовувані архітектури глибинного навчання. У DNN є два недоліки. Вони перенавчаються і мають великий час обчислень. Перенавчання це коли DNN дізнається дуже конкретні деталі навчальних даних за допомогою своїх прихованих шарів. Як результат, DNN працює добре, якщо навчальні дані подаються як вхідні, але погано, коли вхідні дані змінюються. Ця проблема вирішується методом, який називається "виключення" (dropout), коли деякі одиниці випадковим чином видаляються із прихованих шарів під час навчання.

2.1. Нейронні мережі

Робота над штучними нейронними мережами, була мотивована визнанням того, що людський мозок обчислює зовсім не так, як звичайний цифровий комп'ютер. Мозок — це дуже складний, нелінійний і паралельний комп'ютер. Він має можливість впорядковувати свої структурні складові, відомі як нейрони, щоб виконувати певні обчислення у рази швидше, ніж найшвидший цифровий комп'ютер, який існує сьогодні. Мозок регулярно виконує завдання розпізнавання сприйняття приблизно за 100-200 мс, тоді як завдання набагато меншої складності можуть зайняти дні на звичайному комп’ютері.

Народжуючись, мозок має чудову структуру і здатність вибудовувати власні правила за допомогою того, що ми зазвичай називаємо "досвідом". Дійсно, досвід накопичується з часом, найбільший розвиток мозоку відбувається протягом перших двох років від народження, але триває все життя. «Розвивається» нейрон є синонімом пластичного мозку: пластичність дозволяє нервовій системі, що розвивається, адаптуватися до навколишнього середовища. Подібно до того, як пластичність видається важливою для функціонування нейронів як одиниць обробки інформації в мозку людини, так це відбувається і з нейронними мережами, що складаються зі штучних нейронів. У найзагальнішому вигляді нейронна мережа - це машина, яка призначена для моделювання способу, яким мозок виконує певне завдання чи функцію, що цікавить; мережа зазвичай реалізується за допомогою електронних компонентів

або моделюється в програмному забезпеченні на цифровому комп’ютері. Для досягнення гарної продуктивності нейронні мережі використовують взаємозв'язок простих обчислювальних комірок, що називаються "нейронами" або "процесорними блоками". Таким чином, ми можемо запропонувати таке визначення нейронної мережі, яка розглядається як адаптивна машина ":

Нейронна мережа - це масово паралельно розподілений процесор, що складається з простих блоків обробки, який має природну схильність зберігати досвідчені знання та робити їх доступними для використання. Це нагадує мозок у двох відношеннях:

1. Знання набуваються мережею із навколишнього середовища в процесі навчання.

2. Міцність міжнейронного зв’язку, відома як синаптична вага, використовується для зберігання набутих знань.

Процедура, що використовується для виконання навчального процесу, називається алгоритмом навчання, функція якого полягає у впорядкованому зміні синаптичних ваг мережі для досягнення бажаної цілі проектування. Модифікація синаптичних ваг забезпечує традиційний метод проектування нейронних мереж. [4]

Основні розв'язувані завдання штучних нейронних мереж:

- класифікація
- апроксимація
- оптимізація
- кластеризація

2.2 Біологічний нейрон

Нейрони є основними функціональними одиницями нервової системи, вони генерують електричні сигнали, звані потенціалами дії, що дозволяє їм швидко передавати інформацію на великі відстані. Майже всі нейрони виконують три

основні функції необхідні для нормального функціонування всіх клітин організму.

Вони повинні:

1. Отримувати сигнали (або інформацію) ззовні.

2. Обробляти вхідні сигнали та визначати, чи слід передавати інформацію.

3. Повідомляти сигнали потрібним клітинам, якими можуть бути інші нейрони, м’язи або залози.

А тепер давайте розберемося в основних частинах нейрона, щоб глибше зрозуміти, як вони насправді працюють.

Біологічний нейрон складається з 3 основних частин і зовнішньої частини, званої синапсом:

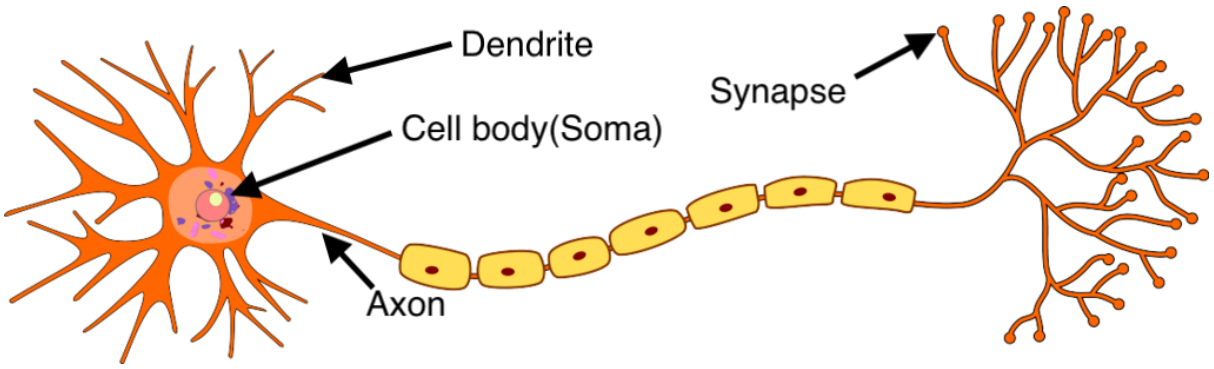


Рисунок 2.2 – Структура біологічного нейрону

Дендрит

Дендрити відповідають за отримання вхідних сигналів ззовні

Сома

Сома - це тіло нервових клітин, відповідальне за обробку вхідних сигналів і рішення, чи повинен нейрон спрацьовувати вихідний сигнал

Аксон

Аксон відповідає за отримання оброблених сигналів від нейрона до відповідних клітин

Синапс

Синапс - це зв’язок між аксоном та іншими дендритами нейронів

Робота цих складових

Завдання отримання вхідної інформації виконується дендритами, і обробка, як правило, відбувається в тілі клітини. Вхідні сигнали можуть бути або збудливими - це означає, що вони, як правило, провокують нейрон (він генерує електричний імпульс) - або гальмівними - що означає, що вони, як правило, не дають нейрону генерувати імпульс.

Більшість нейронів отримують багато вхідних сигналів у своїх дендритних деревах. Один нейрон може мати більше одного набору дендритів і може приймати багато тисяч вхідних сигналів. Незалежно від того, збуджується нейрон чин ні, імпульс залежить від суми всіх збудливих та гальмівних сигналів, які він отримує. Обробка цієї інформації відбувається в сомі, яка є тілом нейронних клітин. Якщо нейрон все ж таки генерує імпульс, нервовий імпульс або потенціал дії проводиться вниз по аксону.

Ближче до свого кінця аксон розпадається на безліч гілок і розвиває цибулинні набряки, відомі як закінчення аксонів (або нервові закінчення). Ці термінали аксонів встановлюють з'єднання на клітинах-мішенях.

2.3 Штучний нейрон

Штучний нейрон, також відомий як персептрон, є основною одиницею нейронної мережі. Простіше кажучи, це математична функція, заснована на моделі біологічних нейронів.

Кожен штучний нейрон виконує такі основні функції:

1. Бере вхідні дані з шару входу
2. Зважує їх окремо і підсумовує

3. Передає суму через нелінійну функцію для отримання результату

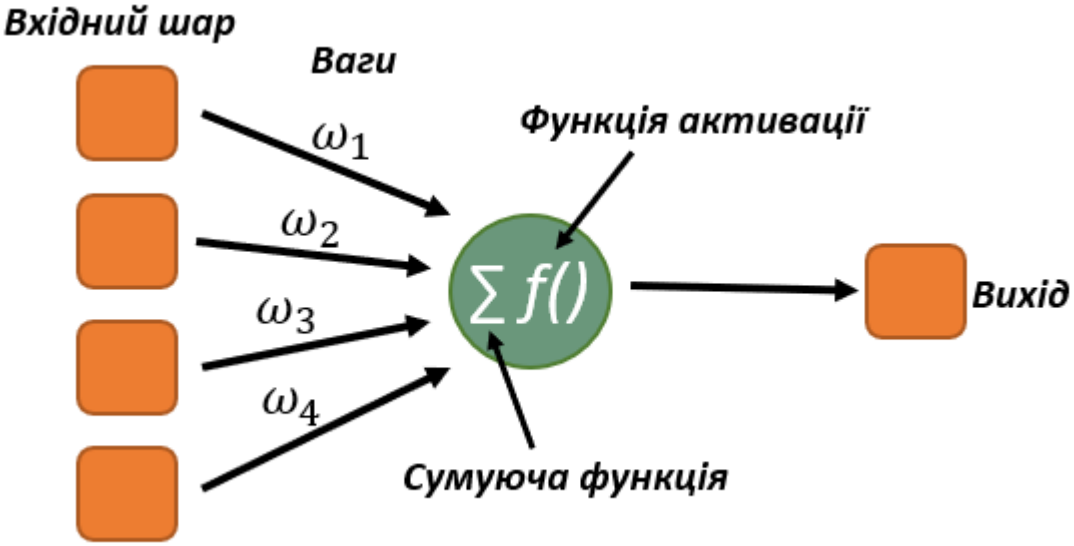


Рисунок 2.3 – Проста структура персептона

Персептрон складається з 4 частин:

1. Вхідні значення або один вхідний шар

За допомогою цього шару ми передаємо вхідні значення нейрону. Це може бути щось таке просте, як колекція значень масиву. Він схожий на дендрит в біологічних нейронах.

2. Ваги та упередження

Ваги - це сукупність значень масиву, які множаться на відповідні вхідні значення. Потім ми беремо суму всіх цих помножених значень, яка називається зваженою сумою. Далі ми додаємо значення упередження до зваженої суми, щоб отримати остаточне значення для прогнозування нашим нейроном.

3. Функція активації

Функція активації вирішує, чи запускається нейрон. Вона вирішує, яке з двох вихідних значень має генерувати нейрон.

4. Вихідний шар

Вихідний шар дає кінцевий вихід нейрона, який потім може бути переданий іншим нейронам мережі або прийнятий як кінцеве вихідне значення.

Розглянемо роботу штучного нейрона на прикладі.

Розглянемо нейрон з двома входами (x1, x2), як показано нижче:

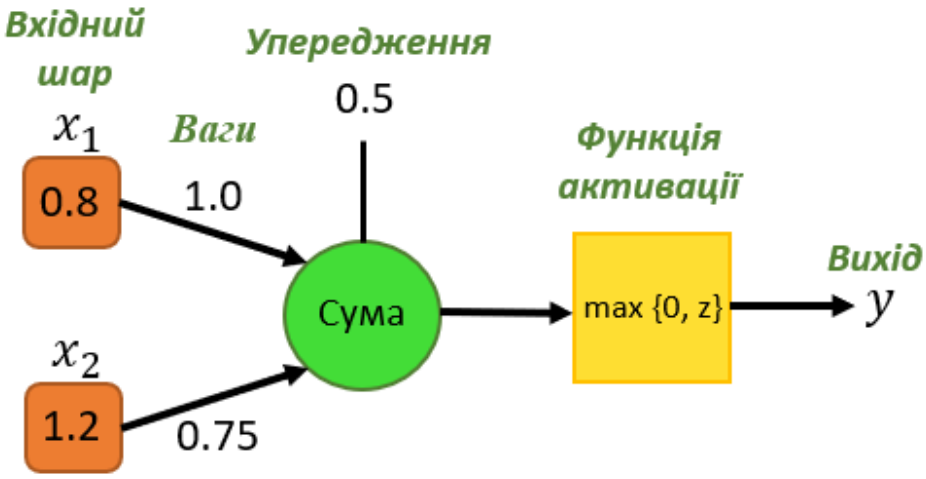


Рисунок 2.4 – Приклад одношарового нейрона

Значення двох входів (x1, x2) становлять 0,8 та 1,2

У нас є набір вагових коефіцієнтів (1.0 та 0.75), що відповідають двом значенням входу

Тоді ми маємо зміщення(bias) зі значенням 0,5, яке потрібно додати до суми

Функція активації обчислюється за формулою:

$$C = w_1 \times x_1 + w_2 \times x_2 = (1 \times 0.8) + (0.75 \times 1.2) + 0.5 = 2.2$$

Тепер комбінація (C) може подаватися до функції активації. Давайте спочатку поглянемо логіку функції ReLU (Rectified linear) активації, яку ми зараз використовуємо в нашому прикладі:

$$activation = \begin{cases} 0 & if \quad combination < 0 \\ combination & if \quad combination \geq 0 \end{cases}$$

Рисунок 2.5 – функція активації ReLU

У нашому випадку отримане нами значення комбінації становило 2,2, що перевищує 0, тому вихідне значення нашої функції активації буде 2,2.

Це буде остаточне вихідне значення нашого одношарового нейрона. [5]

Біологічний нейрон	Штучний нейрон
Клітинне ядро (Сома)	Нейрон
Дендрити	Вхідні дані
Синапс	Ваги
Аксон	Вихідні дані

Таблиця 2.1 – Відповідність між складовими біологічного нейрону та штучного

Коли нейрони з'єднати разом, ми отримаємо нейронну мережу. У нас є один вхідний рівень, один вихідний шар і певна кількість прихованих шарів. Якщо нейронна мережа має лише 1 прихований шар, вона називається тіньовою NN. Якщо нейронна мережа має більше одного прихованого шару, вона називається глибинною нейронною мережею.

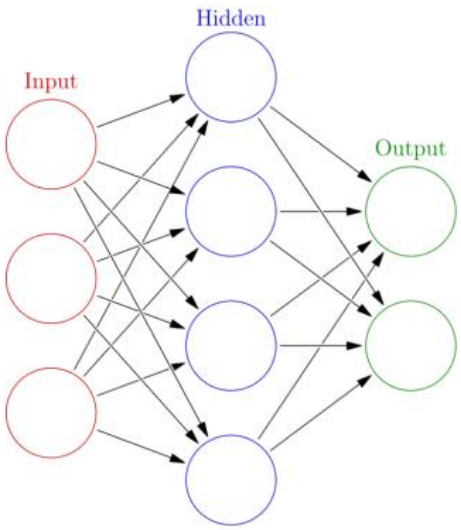


Рисунок 2.6 – Структура зв'язку нейронної мережі

Функціїї активації

Функція одиничного нейрона виражається за допомогою функції активації нейронів (Neuron Activation Function). Похідна функції активації нейронів (DNAF) використовується на етапі навчання нейронних мереж. Як свідчать дані, існують різні типи функцій активацій нейронних мереж, серед них: ReLu, сигмовидні, експоненціальні, порогові тощо. Згідно з дослідженнями, найчастіше використовуються сигмоїди.

Сигмоїди знаходять своє застосування в нейронних мережах для внесення нелінійності в їх роботу з мінімальними змінами в кінцевих результатах. Важливою перевагою сигмоїдної функції у порівнянні з іншими є її досить просте вираження (це дало можливість зменшити складність обчислення ‘’backpropagation algorithm’’).

До класу сигмоїд відносяться:

- $f(s) = \frac{1}{1+e^{-2\alpha s}}$ (експоненційна сигмоїда)
- $f(s) = \frac{s}{|s|+\alpha}$ (раціональна сигмоїда)
- $f(s) = th\frac{s}{\alpha} = \frac{e^{\frac{s}{\alpha}} - e^{-\frac{s}{\alpha}}}{e^{\frac{s}{\alpha}} + e^{-\frac{s}{\alpha}}}$ (гіперболічний тангенс)

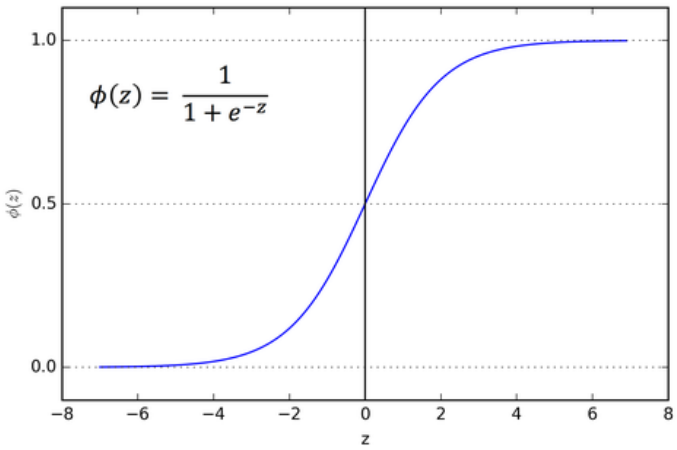


Рисунок 2.7 – Сигмовидна функція

Оскільки, властивістю сигмоїди є «згладжування» значень, то з’являється можливість точного наближення функції до багатьох змінних, використовуючи

лінійні функції, суперпозицію функцій, а також такі операції, як множення та додавання чисел.

Зазвичай, елементи напівавтоматичного навчання використовувати диференційні підсилювачі для реалізації схем за допомогою характеристик сигмовидної функції активації нейронів.

Наступною, але не менш важливою, є активаційна функція ReLu:

$A(x) = max(0, x)$

ReLu є лінійною в першому квадранті, але ReLu нелінійна за своєю природою, так само як і комбінація ReLu.

Дана функція є хорошим апроксиматором, оскільки будь-яка функція може бути апроксимована комбінацією ReLu. З цього випливає, що у нас є можливість зливати шари. Область допустимих значень ReLu - [0, inf).

Іншим плюсом цієї функії є «розрідженість». При використання сигмоїд обов'язковою є активація всіх нейронів. З чого випливає, що майже всі активації повинні бути обробленими для опису виходу мережі. Це був приклад «щільної» активації, вона є трудомісткою та затратною. Для збільшення ефективності активації потрібно, щоб деякі нейрони не були активовані, відповідно активації будуть «розрідженими». У ReLu включається менша кількість нейронів, що дає можливість мережі стати легшою.

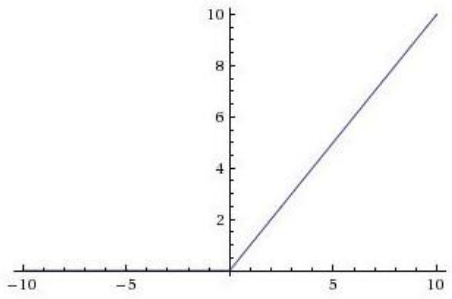


Рисунок 2.8 – Функція ReLu

Але у цієї функції є і мінуси. Основним є проблема вмираючого ReLu (Dying ReLu problem). У зв'язку з цим, коли відбувається спуск, нейрони, які знаходяться у такому стані, не відповідають на вхідні дані або помилки, бо

градієнт дорівнює нулю. Суттю проблеми є те, що деякі нейрони просто вимикаються і не відповідають, роблячи значну частину нейромережі пасивною. Однак існують варіації ReLu, які допомагають уникнути цієї проблеми (можна замінити горизонтальну частину функції на лінійну).

Варто зазначити, що ReLu менш вимогливо відноситься до обчислювальних ресурсів, ніж, наприклад, сигмоїда, так як проводить більш прості математичні операції. Дану функцію використовують для створення глибинних нейронних мереж.

Чому працюють нейронні мережі

Весь секрет роботи нейронних мереж полягає в роботі синапсів. Синапси - місце стику виходу одного нейрона і входу іншого, де відбувається посилення і ослаблення сигналу. У посиленні і ослабленні сигналу і відбувається вся суть роботи і навчання нейронних мереж. Якщо при навчанні правильно підібрати параметри в синапсах, то вхідний сигнал, після проходу через нейронну мережу, буде перетворюватися в вірний сигнал на виході. [6]

РОЗДІЛ 3: Згорткова нейронна мережа

Переваги Згорткових мереж над багатошаровими полягають у використанні загальних ваг в згортальних шарах, що означає, що для кожного пікселя згорткового шару використовується один і той же фільтр.

Згорткова нейронна мережа(CNN) - це конвекційна ANN, яка є дуже ефективною при виконанні завдань розпізнавання та класифікації зображень. CNN нагадує людський спосіб розпізнавання. Вона приймає пікселі як вхідні зображення, перетворює їх за допомогою математичних обчислень та функції активації, а потім створює класифікатор для вхідних класів. CNN має специфічну для кодування особливість у своїй архітектурі, яка підходить для виконання завдань з обробки зображень та комп'ютерного зору. CNN застосовується в таких областях, як аналіз відео, розпізнавання зображень, обробка природних мов (NLP), аналіз ризиків для здоров'я. Архітектура CNN складається з трьох основних шарів, а саме - згорткового шару, агрегувального шару та повноз'єднального шару.

CNN використовують порівняно мало попередньої обробки, в порівнянні з іншими алгоритмами класифікування зображень. [7]

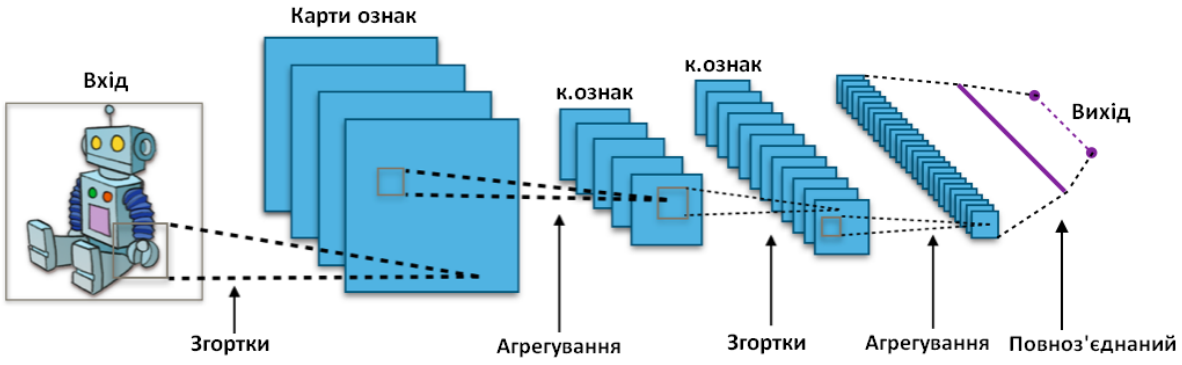


Рисунок 3.1 – Архітектура CNN

CNN автоматично виділяють потрібні дані з вхідних зображень чи відео для оптимальної побудови нейронної мережі. До нейронної мережі додаються додаткові шари згорток, які покликані виділити дані ознаки сильніше на фоні

всього іншого перед подачею даних. Вхідні дані визначаються як тензор другого або третього рангу, в залежності від кількості каналів, які присутні у даних. Канали це кількість змінних, якими описується колір пікселя. Наприклад, RGB-кодування має 3 канали. Проте для зменшення об’єму даних доволі часто вхідні дані переводять в один чорно-білий канал. Тому наступні формули ми будемо наводити у припущенні, що ми працюємо з чорно-білим зображення розміру $n \times m$

Згортковий шар

Згортковий шар — це перший і важливий сегмент архітектури CNN, який фокусується на використанні ядер або параметрів фільтра вхідного зображення. Ядра невеликого розміру і поширюються розмірно на глибину вхідних параметрів. Як тільки вхідні параметри досягають згорткового шару, шар розподіляє кожен піксель по вхідній розмірності, і відображає їх на дві напрямні карти за допомогою функції активації. Наприклад, зображення розміром 5 x 5 пікселів, згортковий шар множить матрицю зображення з матрицею характеристик (3 x 3), яка називається картою об'єктів і видає вихід, представлений нижче.

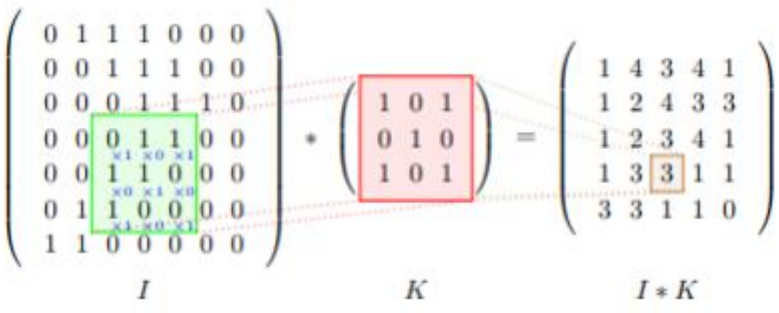


Рисунок 3.2 – перетворення згорткою

Агрегуючий шар

При використанні згорткового шару ми збільшуємо дані у кілька разів. Швидкість роботи та навчання моделі стає проблемою і саме тому доволі часто згортковий шар використовують разом із агрегуючим.

У результаті агрегуючого шару кількість даних зменшується в k 2 разів, де k – розмір вікна, по якому проводиться агрегування. Агрегування шляхом усереднення вікна та шляхом максимимізації вікна - два найбільш поширені способи агрегування. Для обчислення даних шарів використовується вікно. Зображення розбивається на дані вікна, які при цьому не перетинаються і перекривають максимальну площу зображення. В межах кожного вікна обирається максимальне або відповідно середнє значення.

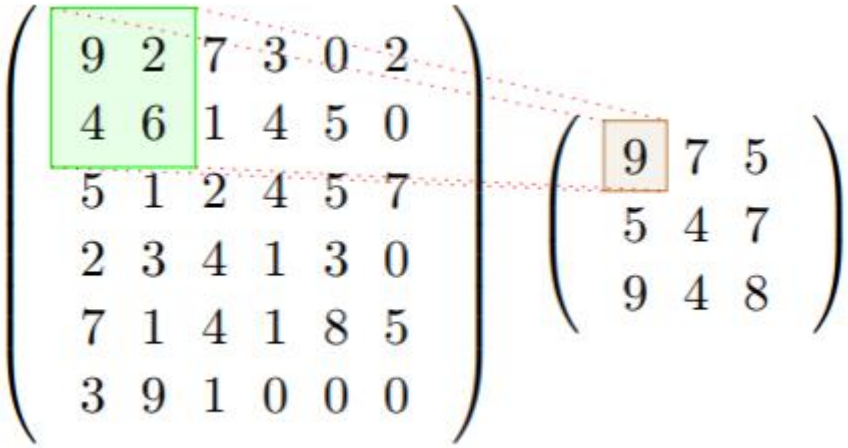


Рисунок 3.3 – Агрегування максимізацією(max pooling)

Повноз'єднальний шар

Повноз'єднальний шар - це останній шар архітектури CNN, який згладжує вихідні дані матриці об’єднання та подає його до мережі нейронів. Нейрони взаємопов’язані і утворюють штучну нейронну мережу. У цьому шарі є сигмоїдні функції активації та SoftMax, що узагальнює весь процес класифікації зображення.

Класифікація зображень із використанням згорткової нейронної мережі - це процес прямого поширення, який є наступним процесом :

Спочатку подаються розміри зображення (пікселі) у згортковий шар.

- Правильно обираються параметри зображення та згортка зображення за допомогою карти активації.
- Застосування агрегування, щоб мінімізувати розмірність зображення.
- Додавання більше згорткових шарів, поки модель не досягне найкращої бажаної точності.
- Згладжування результату і подання в останній шар - повноз'єднальний шар
- Отримання результату, використовуючи функцію активації, яка класифікує вхідне зображення у відповідний клас.

Архітектури згорткових нейронних мереж

У цьому розділі ми розглянемо кілька широко використовуваних архітектур згорткових нейронних мереж, які використовуються сьогодні. Ці мережеві архітектури використовуються не тільки для класифікації, але також, з незначними змінами, використовуються для сегментації, локалізації та виявлення. Крім того, існують попередньо навчені версії кожної з цих мереж, які дають змогу спільноті навчатись трансферу або вдосконалювати моделі.

VGG16

Група VGG у 2014 році стала другою в конкурсі ILSVRC-2014 з 16-шаровою архітектурою під назвою VGG16. Вона використовує глибоку, але просту архітектуру, яка з тих пір набула великої популярності. Замість використання великого розміру фільтра ядра для згортки, архітектура VGG16 використовувала фільтри 3x3, а потім функцію активації ReLU і агрегування максимізацією з 2x2 рецептивним полем. Аргументи винахідників полягали в тому, що використання двох шарів згортки 3x3 еквівалентно наявності однієї згортки 5x5 при збереженні переваг меншого розміру фільтра ядра; тобто реалізація зменшення кількості параметрів та реалізація більшої нелінійності через дві пари згортки – ReLU на противагу одній. Особливою властивістю цієї мережі є те, що, оскільки

просторові розміри вхідного обсягу зменшуються внаслідок згортки та максимального об’єднання, кількість карт функцій збільшується за рахунок збільшення кількості фільтрів, коли заглиблюємось у мережу.

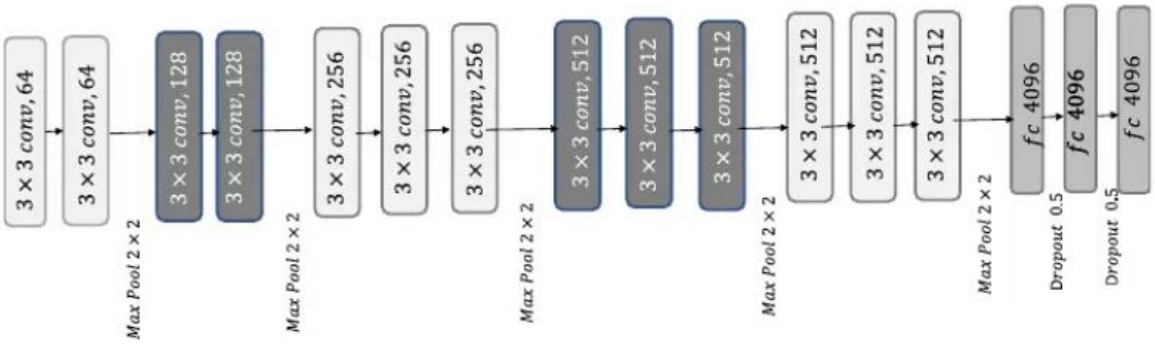


Рисунок 3.4 – Архітектура VGG16

На малюнку представлена архітектура VGG16. Вхідними даними в мережу є зображення розміром 224x224x3. Перші два згорткові шари створюють 64 карти функцій, кожна з яких має агрегування максимізацією після. Фільтри для згортки мають просторовий розмір 3x3. Агрегування максимізацією розміром 2x2 з кроком 2 для всієї мережі. Третій і четвертий згорткові шари створюють 128 функціональних карт, кожна з яких має шар агрегування. Решта мережі слід аналогічним чином, як показано на малюнку. В кінці мережі є три повно'зєднальних шари по 4096 одиниць, за кожним з яких слідує шар SoftMax з тисячі класів. Для повно'зєднальних шарів регуляризація встановлена на 0,5. Усі блоки в мережі мають функцію активацію ReLU.

ResNet

ResNet - це 152-шарова згорткова нейронна мережа від Microsoft, яка перемогла у конкурсі ILSVRC 2015 із коефіцієнтом помилок лише 3,6 відсотка, що, як вважають, є кращим за рівень людських помилок 5–10 відсотків. Крім того, що ця архітектура є глибинною, ResNet реалізує унікальну ідею залишкового блоку. Після кожної серії операцій згортки - ReLU, вхідні дані операції повертається назад на вихід результат операції. Традиційними методами, виконуючи згортку та інші перетворення, ми намагаємося пристосувати основне відображення до

вихідних даних для вирішення завдання класифікації. Однак, використовуючи концепцію ResNet для залишкового блоку, ми намагаємося вивчити залишкове відображення, а не пряме відображення від входу до виходу. Формально в кожному невеликому блоці дій ми додаємо вхід до блоку до виходу. Це проілюстровано на малюнку. Ця концепція базується на гіпотезі про те, що залишкове відображення легше скласти, ніж відповідати оригінальному відображенню від входу до виводу.

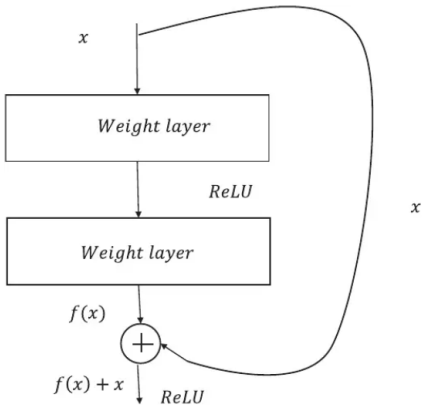


Рисунок 3.5 – Залишковий блок

Передавальне навчання (Transfer Learning)

Передавальне навчання (Transfer Learning) в широкому розумінні означає зберігання знань, отриманих під час вирішення проблеми, та використання цих знань для іншої проблеми в подібній області. Навчання за допомогою трансферу мало величезний успіх у галузі глибинного навчання з різних причин. Моделі глибинного навчання загалом мають величезну кількість параметрів через природу прихованих шарів та схем зв’язку в різних блоках. Щоб навчити таку величезну модель, потрібно багато даних, або модель страждатиме від проблем із перенавчанням. У багатьох проблемах величезний обсяг даних, необхідних для навчання моделі, недоступний, але природа проблеми вимагає глибинного рішення, щоб мати найкращий результат. Наприклад, при обробці зображень для розпізнавання об’єктів моделі глибинного навчання забезпечують найсучасніші

рішення. У таких випадках навчання за допомогою трансферу може бути використано для генерації загальних ознак із заздалегідь підготовленої моделі глибокого навчання, а потім використовувати ці функції для побудови простої моделі для вирішення проблеми. Отже, єдиними параметрами для цієї задачі будуть ті, що використовувались для побудови простої моделі. Попередньо навчені моделі, як правило, навчаються на величезному масиві даних і тому мають надійні характеристики. Коли ми обробляємо зображення через кілька згорткових шарів, початкові шари вчаться виявляти дуже загальні ознаки, такі як криві та краї. По мірі того, як мережа стає глибшою, згорткові шари в глибоких шарах вчаться виявляти більш складні ознаки, що стосуються конкретного виду набору даних. Наприклад, у класифікації глибинні шари навчаються виявляти такі особливості, як очі, ніс, обличчя тощо. Припустимо, у нас є модель архітектури VGG16, навчена на тисячі категорій набору даних. Тепер, якщо ми отримаємо менший набір даних, який має менше категорій зображень, подібних до тих, що є у набору даних моделей, підготовлених VGG16, тоді ми можемо використовувати ту саму модель VGG16 аж до повноз'єднаного шару, а потім замінити вихідний шар новими класами. Крім того, ми зберігаємо ваги мережі фіксованими до повноз'єднаного шару і лише тренуємо модель для вивчення ваг від повноз'єднаного шару до вихідного. Це пов'язано з тим, що природа набору даних така ж, як і меншого набору даних, і, отже, особливості, засвоєні в попередньо навченій моделі за допомогою різних параметрів, є достатніми для нової проблеми класифікації, і нам потрібно лише вивчити ваги з повністю зв'язаних шару до вихідного шару. Це величезне зменшення кількості параметрів, які потрібно вивчити, і це зменшить переобладнання. Якби ми навчили малий набір даних за допомогою архітектури VGG16, він міг постраждати від серйозного перенавчання через велику кількість параметрів для вивчення на малому наборі даних. Якщо природа набору даних сильно відрізняється від набору даних, який використовувався для попередньо навченої моделі, ми можемо використовувати ту саму попередньо навчену модель, але виправити лише параметри для першої пари наборів згортки – ReLUs –

агрегування максимізацією шарів, а потім додати пару згорток – ReLU – агрегування максимізацією шарів, які навчаться виявляти особливості, властиві новому набору даних. Нарешті, ми повинні мати повноз'єднаний шар, за яким слідіє вихідний шар. Оскільки ми використовуємо ваги початкових наборів шарів згортки – ReLUs з попередньо навченої мережі VGG16, параметри щодо цих шарів вивчати не потрібно. Як вже згадувалося раніше, ранні шари згортки вивчають дуже загальні особливості, такі як краї та криві, що застосовуються до всіх видів зображень. Решту мережі потрібно було б навчити вивчати специфічні особливості, властиві конкретному набору проблем [8]

РОЗДІЛ 4: Опис реалізації програмного застосунку

Обґрунтування вибору засобів розробки

У якості засобу розробки було використано мову Python, адже це найпопулярніша мова для задач машинного навчання через велику кількість фреймворків та бібліотек, які спрощують процес написання коду і скорочують час на розробку. Середовищем програмування було обрано Kaggle Notebook. Це безкоштовний хмарний сервіс на основі Jupyter Notebook та платформи Kaggle. Він надає все необхідне для машинного навчання прямо в браузері та зручне завантаження датасету. Для наглядної презентації різних архітектур навчання нейронних мереж був обраний датасет зображень фруктів та овочів, який називається Fruits 360 з платформи Kaggle.

Параметри датасету:

- Загальна кількість зображень: 90483.
- Розмір тренувального датасету: 67692 картинок
- Розмір тестувального датасету: 22688 картинок
- Кількість класів: 131
- Розмір зображень: 100x100 пікселів.

Python

При моделюванні нейронних мереж, використовується мова програмування Python. Python – гнучка мова програмування високого рівня, створена в 1991 році. Спеціаліст з машинного навчання повинен збирати, систематизувати і аналізувати дані, а потім на основі отриманої інформації створювати алгоритми для штучного інтелекту, а Python найкраще підходить для виконання таких завдань, тому що він досить зрозумілий в порівнянні з іншими мовами. Більш того, у нього відмінна продуктивність при обробці даних. Він дуже простий в освоєнні, крім того, нейронні мережі створюють і навчають в основному на цій мові. Одна з основних причин, чому Python використовується для машинного

навчання полягає в тому, що у нього є безліч фреймворків, які спрощують процес написання коду і скорочують час на розробку. [10]

Kaggle

Kaggle, дочірнє підприємство Google LLC, дозволяє користувачам знаходити та публікувати набори даних, досліджувати та будувати моделі у веб-середовищі науки про дані, співпрацювати з іншими вченими даних та інженерами машинного навчання та брати участь у змаганнях для вирішення проблем науки про дані. Kaggle розпочав свою діяльність у 2010 році, пропонуючи змагання з машинного навчання, а тепер також пропонує публічну платформу даних, хмарний робочий стіл для науки про дані та освіту з питань штучного інтелекту.

Kaggle Notebook це більше, ніж просто редактор коду. Це версійне обчислювальне середовище, розроблене для спрощення відтворення роботи з наукою про дані. В IDE мається доступ до інтерактивного сеансу, що працює в контейнері Docker із попередньо встановленими пакетами, можливість монтувати версійні версії джерел даних, настроювані обчислювальні ресурси, такі як графічні процесори тощо. [11]

NumPy

NumPy - це основний пакет для наукових обчислень у Python. Це бібліотека Python, яка забезпечує багатовимірний об’єкт масиву, різні похідні об’єкти (наприклад, масковані масиви та матриці) та асортимент підпрограм для швидких операцій над масивами, включаючи математичні, логічні, маніпуляції з фігурами, сортування, вибір, введення / виведення , дискретні перетворення Фур'є, основна лінійна алгебра, основні статистичні операції, випадкове моделювання та багато іншого. [12]

TensorFlow

TensorFlow - це фреймворк глибокого навчання з відкритим кодом від Google, натхненний Theano. TensorFlow поступово стає найкращою бібліотекою для глибокого навчання в дослідницькій роботі, а також для впровадження у виробництві. Завдяки впровадження у хмарі, TensorFlow стає ідеальною

бібліотекою для машинного навчання. Інша програма Keras виконується поверх TensorFlow для запуску різних програм. Реалізація нейронних мереж є не скадним процесом, завдяки цим двом. Зв'язок TensorFlow з Гуглом допомогли проекту залучити багато уваги. Основна бібліотека підходить для широкого сімейства технік машинного навчання, а не тільки для глибинного навчання. На додаток до основної функціональності машинного навчання, TensorFlow також включає власну систему логування, власний інтерактивний візуалізатор логів і потужну архітектуру з доставки даних. Модель виконання TensorFlow відрізняється від scikit-learn мови Python і від більшості інструментів в R.

Розробка програмного продукту

Обробка обраного датасету

Імпортування необхідних бібліотек:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.datasets import load_files
from keras.utils import np_utils
from keras.preprocessing.image import array_to_img, img_to_array, load_img
import matplotlib.pyplot as plt

import os
```

Крім засобів розробки зазначених вище, необхідні бібліотеки включають в себе:

Keras – відкрита нейромережна бібліотека, написана мовою Python

Sklearn – бібліотека, у якій реалізовані методи машинного навчання мовою Python та використовує numpy для високопродуктивних операцій лінійної алгебри та операцій з масивами. Має функціонал для створення та тренування різноманітних алгоритмів кластеризації, регресії та класифікації.

Matplotlib — бібліотека на мові програмування Python для візуалізації даних двовимірною 2D графікою

Pandas – бібліотека, яка створює масиви даних та інструменти керування цими даними реалізована мовою Python

Шлях обраного датасету

In [4]:
```
train_dir = "/kaggle/input/fruits/fruits-360/Training"
test_dir = "/kaggle/input/fruits/fruits-360/Test"
```

Користуючись Kaggle Notebook маємо дуже швидкий та легкий спосіб завантаження датасету.

Вивід розміру тренувальноо та тестувального датасету та кількості класів:

In [5]:
```
def load_dataset(path):
    data = load_files(path)
    files = np.array(data['filenames'][:20000])
    targets = np.array(data['target'][:20000])
    target_labels = np.array(data['target_names'])
    return files,targets,target_labels

x_train, y_train,target_labels = load_dataset(train_dir)
x_test, y_test,_ = load_dataset(test_dir)
print('Loading complete!')

print('Training set size : ' , x_train.shape[0])
print('Testing set size : ', x_test.shape[0])
```

```
Loading complete!
Training set size :  20000
Testing set size :  20000
```

In [6]:
```
no_of_classes = len(np.unique(y_train))
no_of_classes
```

Out[6]:
```
131
```

Перетворення масиву мічених даних (від 0 до nb_classes - 1) в унітарний код та розподіл датасету на train, test та validation блоки, де 50% тренувальний, 17.5% валідація і 32.5% тестовий:

```
In [7]:
y_train = np_utils.to_categorical(y_train,no_of_classes)
y_test = np_utils.to_categorical(y_test,no_of_classes)
y_train[0]
```

```
Out[7]:
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

```
In [8]:
x_test,x_valid = x_test[7000:],x_test[:7000]
y_test,y_vaild = y_test[7000:],y_test[:7000]
print('Vaildation X : ',x_valid.shape)
print('Vaildation y :',y_vaild.shape)
print('Test X : ',x_test.shape)
print('Test y : ',y_test.shape)
```

```
Vaildation X :  (7000,)
Vaildation y : (7000, 131)
Test X :  (13000,)
Test y :  (13000, 131)
```


```
In [7]:
y_train = np_utils.to_categorical(y_train,no_of_classes)
y_test = np_utils.to_categorical(y_test,no_of_classes)
y_train[0]
```

```
Out[7]:
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```


Конвертація зображень в масив:

In [9]:
```
def convert_image_to_array(files):
    images_as_array=[]
    for file in files:
        # Convert to Numpy Array
        images_as_array.append(img_to_array(load_img(file)))
    return images_as_array

x_train_ = np.array(convert_image_to_array(x_train))
del x_train
x_valid_ = np.array(convert_image_to_array(x_valid))
del x_valid
x_test_ = np.array(convert_image_to_array(x_test))
del x_test
```

In [10]:
```
x_train = x_train_.astype('float32')/255
del x_train_
```

In [11]:
```
x_valid = x_valid_.astype('float32')/255
del x_valid_
```

In [12]:
```
x_test = x_test_.astype('float32')/255
del x_test_
```

Найчастіше використовують 32-бітову точність при навчанні нейронної мережі, тому в один момент навчальні дані доводиться перетворити на 32-бітові плаваючі. Щодо ділення на 255, це максимальне значення тож це забезпечить регулювання вхідних функцій від 0,0 до 1,0.

```
In [13]:
fig = plt.figure(figsize =(30,5))
for i in range(10):
    ax = fig.add_subplot(2,5,i+1,xticks=[],yticks=[])
    ax.imshow(np.squeeze(x_train[i]))
```

Fully trained ResNet50

Перша реалізована модель нейронної мережі є ResNet50

50-шарова ResNet: кожен 3-шаровий блок замінюється в 34-шарової мережі цим 3-шаровим вузьким місцем, в результаті виходить 50-шарова ResNet (див. Таблицю вище). Вони використовують варіант 2 для збільшення розмірності. Ця модель має 3,8 мільярда FLOPs.

Імпортування необхідних бібліотек:

```
In [ ]:
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D
from keras.layers import Activation, Dense, Flatten, Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint
from keras import backend as K
```

Модель - це основна структура даних Keras. Моделі Keras визначають спосіб впорядкування шарів. В Keras вони доступні двох типів: послідовна модель Keras та функціональна API Keras. Для реалізації необхідної задачі використовуємо послідовна модель. Це дозволяє нам створювати моделі шар за шаром у послідовному порядку. [13]

Створення моделі, шарів Conv2D, агрегувального шару MaxPooling2D

Компіляція моделі. Додавання оптимізатора, функції втрат та метрик

```
model = Sequential()
model.add(Conv2D(filters = 16, kernel_size = 2,input_shape=(100,100,3),paddi
ng='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters = 32,kernel_size = 2,activation= 'relu',padding='sa
me'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters = 64,kernel_size = 2,activation= 'relu',padding='sa
me'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters = 128,kernel_size = 2,activation= 'relu',padding='s
ame'))
model.add(MaxPooling2D(pool_size=2))

model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(150))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(131,activation = 'softmax'))
model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
print('Compiled!')
```

Кількість фільтрів у першому згортковому шарі Conv2D дорівнює 32, розмір ядра: 2, функція активації: ReLu. Всього модель має три згорткові шари, у яких кількість фільтрів кожен збільшується у два рази.

Після згорткових шарів слідують агрегувальні шари MaxPooling2D для 2D просторових даних. Даний шар зменшує вибірку вхідного подання, приймаючи максимальне значення над вікном, визначеним pool_size для кожного виміру вздовж осі об’єктів.

Виключення — метод регуляризації штучних нейронних мереж, призначений для запобігання перенавчання мережі. Метод значно підвищує швидкість

навчання. Суть методу полягає в тому, що в процесі навчання із загальної мережі випадковим чином виділяється підмережа, для якої здійснюється навчання. Після навчання обраної підмережі випадковим чином обирається нова підмережа і навчання продовжується. Вибір нейронів для підмережі відбувається випадковим чином з ймовірністю, яка називається коефіцієнтом дропаута, що для нашої моделі є 0,3 та 0,4. Більш навчені нейрони отримують більшу вагу.

У якості функції втрат цієї моделі було використано категоральну функцію кроссентропії, адже на в нашосу датасеті більше двох класів та мічені дані подаються в вигляді унітарного коду.

Обраним оптимізатором є RMSProp: адже він ідеально підходить для рекурентних нейронних мереж.

Метрика — це функція, яка використовується для оцінки ефективності вашої моделі. Метричні функції подібні до функцій втрат, за винятком того, що результати оцінки метрики не використовуються при навчанні моделі. В нашому випадку було обрано метрику точності(Accuracy metrics), тобто обчислюється, як часто передбачення дорівнюють міткам.

Задання розміру батча

In [ ]:
```
batch_size = 32

checkpointer = ModelCheckpoint(filepath = 'cnn_from_scratch_fruits.hdf5', verbose = 1, save_bes
t_only = True)

history = model.fit(x_train,y_train,
        batch_size = 32,
        epochs=30,
        validation_data=(x_valid, y_vaild),
        callbacks = [checkpointer],
        verbose=2, shuffle=True)
```

In [ ]:
```
model.load_weights('cnn_from_scratch_fruits.hdf5')
score = model.evaluate(x_test, y_test, verbose=0)
print('\n', 'Test accuracy:', score[1])
```

Не можна пропустити через нейронну мережу разом весь датасета. Тому дані потрібно поділити на партії. Наша модель використовує батч розміром 32.

Зворотний виклик ModelCheckpoint використовується разом із навчанням за допомогою model.fit () для збереження моделі чи ваг з певним інтервалом, щоб модель або ваги можна було завантажити пізніше, щоб продовжити навчання із збереженого стану. Завдяки ньому можна зберегти модель, яка досягла «найкращих показників» на сьогодні, чи зберегти модель наприкінці кожної епохи, незалежно від продуктивності.

Реалізація візуалізації тестового прогнозу:

In [ ]:
```
# Let's visualize test prediction.

y_pred = model.predict(x_test)

# plot a random sample of test images, their predicted labels, and ground trut
h
fig = plt.figure(figsize=(16, 9))
for i, idx in enumerate(np.random.choice(x_test.shape[0], size=16, replace=Fa
lse)):
    ax = fig.add_subplot(4, 4, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_test[idx]))
    pred_idx = np.argmax(y_pred[idx])
    true_idx = np.argmax(y_test[idx])
    ax.set_title("{} ({})".format(target_labels[pred_idx], target_labels[true
_idx]),
                 color=("green" if pred_idx == true_idx else "red"))
```

На цьому навчання моделі ResNet50 закінчилось.

Transfer Learning ResNet50

Наступна модель навчання нейронної мережі також ResNet50, але вже за допомогою передавального навчання, тобто навчання буде складаться з використання особливостей, вивчених на одній задачі та використання їх для нової, подібної задачі. В даному випадку буде використовуватися модель вище: Fully trained ResNet50.

In [14]:

```
import os
from os import listdir, makedirs
from os.path import join, exists, expanduser

from keras import applications
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K
import tensorflow as tf
from keras.callbacks import ModelCheckpoint
```

In [15]:

```
#import inception with pre-trained weights. do not include fully #connected la
yers
inception_base = applications.ResNet50(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = inception_base.output
x = GlobalAveragePooling2D()(x)
# add a fully-connected layer
x = Dense(150, activation='relu')(x)
# and a fully connected output/classification layer
predictions = Dense(131, activation='softmax')(x)
# create the full network so we can train on it
model = Model(inputs=inception_base.input, outputs=predictions)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-app
lications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
```

In [16]:

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

In [23]:

```
#import inception with pre-trained weights. do not include fully #connected la
yers
inception_base = applications.ResNet50(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = inception_base.output
x = GlobalAveragePooling2D()(x)
# add a fully-connected layer
x = Dense(150, activation='relu')(x)
# and a fully connected output/classification layer
predictions = Dense(131, activation='softmax')(x)
# create the full network so we can train on it
model = Model(inputs=inception_base.input, outputs=predictions)

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

model.load_weights('cnn_tl_resnet50.hdf5')
score = model.evaluate(x_test, y_test, verbose=0)
print('\n', 'Test accuracy:', score[1])
```

```
Test accuracy: 0.9665384888648987
```

Після передавального навчання модель досягла високої точності: ≈0,966

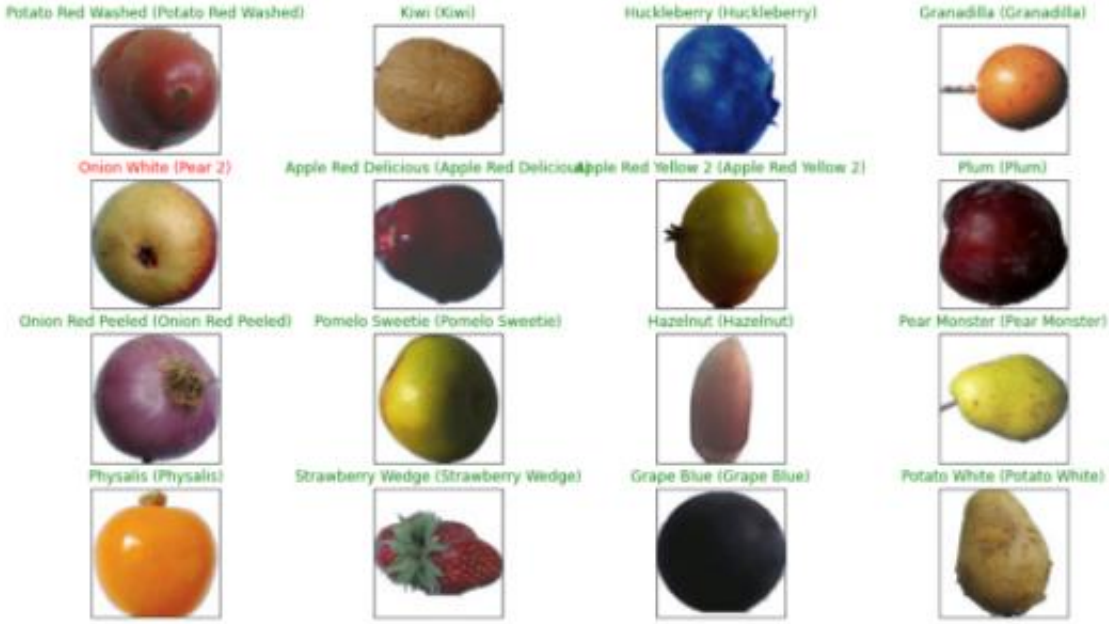
Візуалізація класифікації датасету:

In [24]:

```
# Let's visualize test prediction.

y_pred = model.predict(x_test)

# plot a random sample of test images, their predicted labels, and ground trut
h
fig = plt.figure(figsize=(16, 9))
for i, idx in enumerate(np.random.choice(x_test.shape[0], size=16, replace=Fa
lse)):
    ax = fig.add_subplot(4, 4, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_test[idx]))
    pred_idx = np.argmax(y_pred[idx])
    true_idx = np.argmax(y_test[idx])
    ax.set_title("{} ({})".format(target_labels[pred_idx], target_labels[true
_idx]),
                 color=("green" if pred_idx == true_idx else "red"))
```



Transfer Learning VGG19

Остання третя модель: VGG19. Це дуже гарна модель, адже, по-перше, незважаючи на те, що вона не виграла ILSVRC, вона посіла 2-е місце, показавши хороші результати. По-друге, архітектура VGG19 дуже проста. Якщо ви розумієте базову модель CNN, ви відразу помітите, що VGG19 схожа на базову модель CNN, Часто людей використовують VGG16, тому доречно показати VGG19 модель у дії.

VGG19 - це варіант моделі VGG, яка складається з 19 шарів (16 згорткових шарів, 3 повнозв'язаних шарів, 5 шарів MaxPool та 1 шар SoftMax).

In [25]:

```
#import inception with pre-trained weights. do not include fully #connected la
yers
inception_base = applications.VGG19(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = inception_base.output
x = GlobalAveragePooling2D()(x)
# add a fully-connected layer
x = Dense(256, activation='relu')(x)
# and a fully connected output/classification layer
predictions = Dense(131, activation='softmax')(x)
# create the full network so we can train on it
model = Model(inputs=inception_base.input, outputs=predictions)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-app
lications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
```

In [26]:

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

In [27]:

```
import tensorflow as tf
with tf.device("/device:GPU:0"):
    batch_size = 16
    checkpointer = ModelCheckpoint(filepath = 'cnn_tl_vgg_19.hdf5', verbose =
1, save_best_only = True)
    history = model.fit(x_train,y_train,
            batch_size = 16,
            epochs=30,
            validation_data=(x_valid, y_vaild),
            callbacks = [checkpointer],
            verbose=2, shuffle=True)
```

```
Epoch 1/30
1250/1250 - 63s - loss: 8062.8408 - accuracy: 0.0113 - val_loss: 4.8456 -
val_accuracy: 0.0131

Epoch 00001: val_loss improved from inf to 4.84565, saving model to cnn_t
l_fruits.hdf5
Epoch 2/30
1250/1250 - 59s - loss: 3.7652 - accuracy: 0.1091 - val_loss: 2.8037 - va
l_accuracy: 0.2170
```

```
Epoch 00029: val_loss did not improve from 1.11203
Epoch 30/30
1250/1250 - 59s - loss: 1.0584 - accuracy: 0.8367 - val_loss: 2.9940 - val_accuracy: 0.5976

Epoch 00030: val_loss did not improve from 1.11203
```

In [ ]:

```
# Let's visualize test prediction.

y_pred = model.predict(x_test)

# plot a random sample of test images, their predicted labels, and ground trut
h
fig = plt.figure(figsize=(16, 9))
for i, idx in enumerate(np.random.choice(x_test.shape[0], size=16, replace=Fa
lse)):
    ax = fig.add_subplot(4, 4, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_test[idx]))
    pred_idx = np.argmax(y_pred[idx])
    true_idx = np.argmax(y_test[idx])
    ax.set_title("{} ({})".format(target_labels[pred_idx], target_labels[true
_idx]),
                 color=("green" if pred_idx == true_idx else "red"))
```

Висновки

Використання нейронних мереж для прогнозування та моделювання викликало значний інтерес в останні роки. Швидкі темпи розвитку сучасної науки потребують гарних методів опрацювання інформації, що дозволять заощадити час, потужності та інші ресурси. Нейронні мережі сприяють такому розвитку та полегшують розв'язання багатьох задач. Дана робота аналізує різноманітні методи і моделі машинного навчання та нейронних мереж, їх застосунок у житті.

Було розглянуто область інформатики машинне навчання та його вплив і користь в повсякденному житті.

Порівняння різних моделей навчання нейронних мереж було виконано на прикладі задачі класифікації зображен фруктів із датасету Fruits 360. Було реалізовано моделі Fully-trained ResNet50, Transfer Learning ResNet50 та Transfer Learning VGG19. Програмно модель згорткової нейронної мережі було реалізовано за допомогою відкритої нейромережної бібліотеки Keras мовою Python у середовищі програмування Kaggle Notebook, для виконання складних математичних обчислень використовувалась бібліотека TensorFlow. При передавальному навчанні вдалося значно пришвидшити навчання моделі та досягнуто високої точності. Варто відзначити, що модель ResNet має менше фільтрів і складність менше, ніж мережі VGG, але обидві моделі є чудовими інструментами для вирішування задачі класифікації зображень.

Для вирішення задач машинного навчання мова Python є незамінним інструментом, що забезпечений усіма можливими бібліотеками для полегшення роботи користувача, Kaggle є новою потужною платформою для задач з наукою даних, а бібліотека Keras спроектована для швидких експериментів з мережами глибинного навчання, зручна в користуванні, модульна та розширювальна.

Список використаної літератури

[1] Understanding Machine Learning: From Theory to Algorithms [Електронний ресурс] 2014 Автор Shai Shalev-Shwartz and Shai Ben-David https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf

[2] An Overview of Machine Learning and its Applications 2017 Автори Annina Simon1, Mahima Singh Deo , S. Venkatesan3 , D.R. Ramesh Babu

[3] Machine Learning [Електронний ресурс] Автор brilliant.org https://brilliant.org/wiki/machine-learning/#techniques-in-machine-learning

[4] Neural networks A Comprehensive Foundation, Автор Simon Haykin

[5] What Is The Relation Between Artificial And Biological Neuron? [Електронний ресурс] 2020 Автор Saurabh Mhatre https://towardsdatascience.com/what-is-the-relation-between-artificial-and-biological-neuron-18b05831036

[6] Нейронный сети. Эволюция, Автор Каниа Алексеевич Кан

[7] Deep learning Convolutional Neural Network Image Classification (Python) Computational neuroscience project Автор Victor. O. Owino

[8] Pro Deep Learning with TensorFlow A Mathematical Approach to Advanced Artificial Intelligence in Python Автор Santanu Pattanayak

[9] Fruits 360 [Електронний ресурс] 2020 Автор Mihai Oltean https://www.kaggle.com/moltean/fruits

[10] https://www.python.org/about/apps/

[11] https://www.kaggle.com/docs/notebooks

[12] https://numpy.org/doc/stable/user/whatisnumpy.html

[13] Keras Models – Types and Examples [Електронний ресурс] Автор data-flair.training https://data-flair.training/blogs/keras-models/