Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

Розпізнавання обличчя з використанням нейронних мереж

Текстова частина до магістерської роботи (тези)
за спеціальністю „Інженерія програмного забезпечення”

Виконав: студент 2-го року навчання
Гетьман Максим Сергійович
Керівник Франчук О.В.,
Доцент, кандидат технічних наук
Магістерська робота захищена
з оцінкою «_____»
Секретар ДЕК _____
«____» _____2021 р

Київ 2021

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

доц., канд. фіз.-мат. наук

_____С. С. Гороховський
(підпис)
8 листопада 2020 р

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломну роботу

студенту Гетьману Максиму факультету інформатики 1 курсу МП

ТЕМА: Розпізнавання обличчя з використанням нейронних мереж

Зміст ТЧ до дипломної роботи:

Індивідуальне завдання
Вступ
1. Огляд нейронних мереж
2. Алгоритм розпізнавання обличчя
3. Програмна імплементація алгоритму

Висновки
Список джерел

Дата видачі 8 листопада 2020 р.

Керівник _____

Календарний план виконання дипломної роботи

Тема: Розпізнавання обличчя з використанням нейронних мереж

№ п/п	Назва етапу дипломної роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу	листопад 2020р.	
2.	Огляд літератури за темою роботи	листопад-грудень 2020р.	
3.	Ознайомлення з типами нейронних мереж	грудень 2020р.	
4.	Огляд алгоритмів розпізнавання	січень-лютий 2021р.	
5.	Імплементація такого алгоритму	березень 2021р.	
6.	Тестування та випробування проекту	квітень 2021р.	
7..	Попередній захист	травен 2021р.	
8.	Аналіз та корегування роботи	травень 2021р.	
9.	Створення слайдів для доповіді та написання доповіді.	травень 2021р.	
10.	Остаточне оформлення пояснювальної роботи та слайдів	червень 2021р.	
11.	Захист дипломної роботи	червень 2021р.	

Студент _____Гетьман М.С._____

Керівник _____Франчук О.В..._____

“_____” _____ _____ р.

Зміст

Анотація

У даній дипломній роботі розглянуто нейронні мережі, їхні типи, сфери застосування. Розглянуто алгоритм розпізнавання обличчя з використанням нейронної мережі.

У першому розгляді детально розглянуто що таке нейронна мережа, які є типи та як нейронні мережі навчаються.

У другому розділі детально описано алгоритм розпізнавання обличчя.

У третьому розділі наведена програма імплементація алгоритму із другого розділу.

Вступ

З кожним роком нейронні мережі все більше і більше застосовуються в самих різних сферах. Нові відкриття та ідеї з’являються мало не кожного дня.

Одним із напрямків використання нейронних мереж є розпізнавання зображень, зокрема облич людей.

Таке застосування має прикладний характер - системи безпеки, розблокування мобільного телефону тощо.

Мета даної роботи - дослідити, які існують типи нейронних мереж, розібратися із алгоритмом розпізнавання обличчя та імплементувати таку нейронну мережу.

Розділ 1: Що таке нейронні мережі - історія появи, типи, відмінності

1.1 Біологічні нейрони

Щоб розібратися з поняттям “Нейронні мережі”, спочатку треба зрозуміти, що таке нейрон. Згідно визначення, “Нейрон” - це спеціалізована нервова клітина, що є основною структурною і функціональною одиницею нервової системи. [1]

Його функції:

- Сприйняття подразнень
- Обробка подразнень
- Передача інформації у іншим органам у вигляді нервових імпульсів
- Формування відповідної реакції

Саме завдяки нейронам люди можуть обробляти вхідну інформацію. Нейрони не функціонують самостійно. У живому організмі вони з’єднані між собою, формуючи великі “мережі”. Мозок людини, згідно досліджень, складається приблизно зі ста мільярдів нейронів.[2]

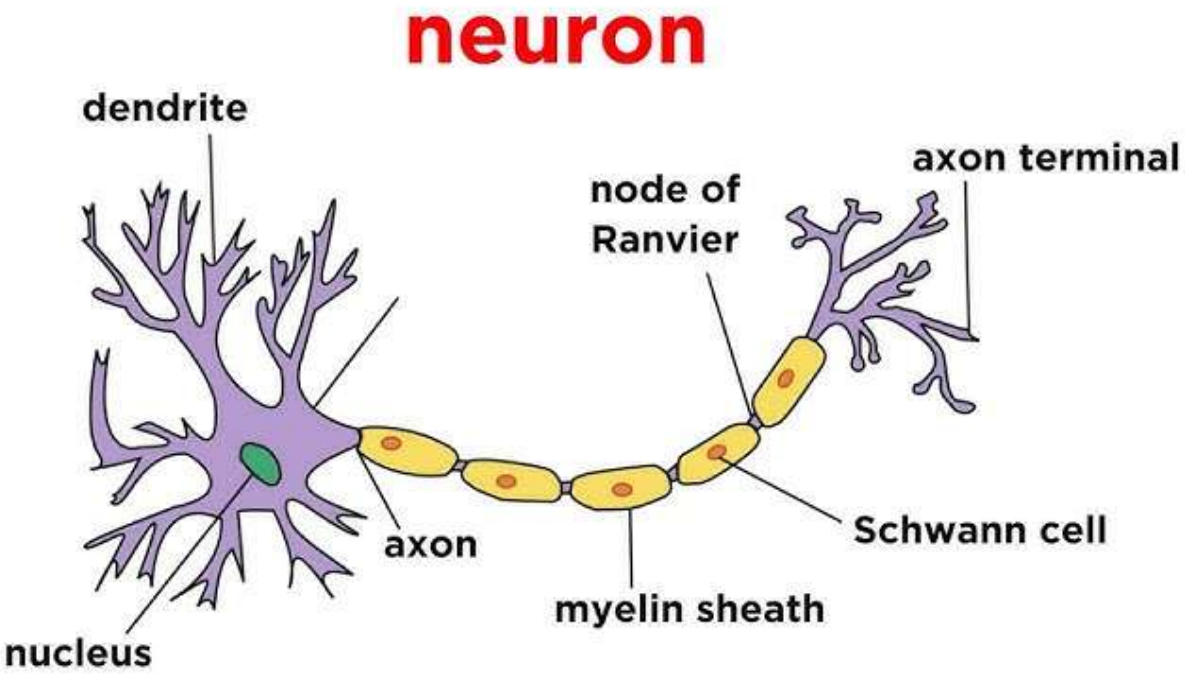

Рис. 1.1

(*https://i.ytimg.com/vi/4RS-3Ex04NU/hq720.jpg?sqp=-oaymwEhCK4FEIIDSFryq4qpAxMIARUAAAAGAElAADIQj0AgKJD&rs=AOn4CLCLXf3SYOr5uQH_ELrSleomOYKLvg)*

1.2 Штучні нейронні мережі

Надихнувшись біологічними нейронними мережами, люди спробували реалізувати схожу модель обчислень завдяки програмному забезпеченню.

Штучні нейронні мережі (англ. *Artificial Neural Networks, ANN*, *нейронні мережі*) – це обчислювальні системі, які симулюють спосіб аналізу та обробки інформації людським мозком.[3]

Перші кроки в створенніх нейронних мереж зробили Воррен Маккалох та Уолтер Піттс в 1943 році. Вони описали теорію, як могли б працювати

нейронні мережі та змоделювали це, використовуючи електричні схеми. [4]. Приблизна схема показана на Рис 1.2

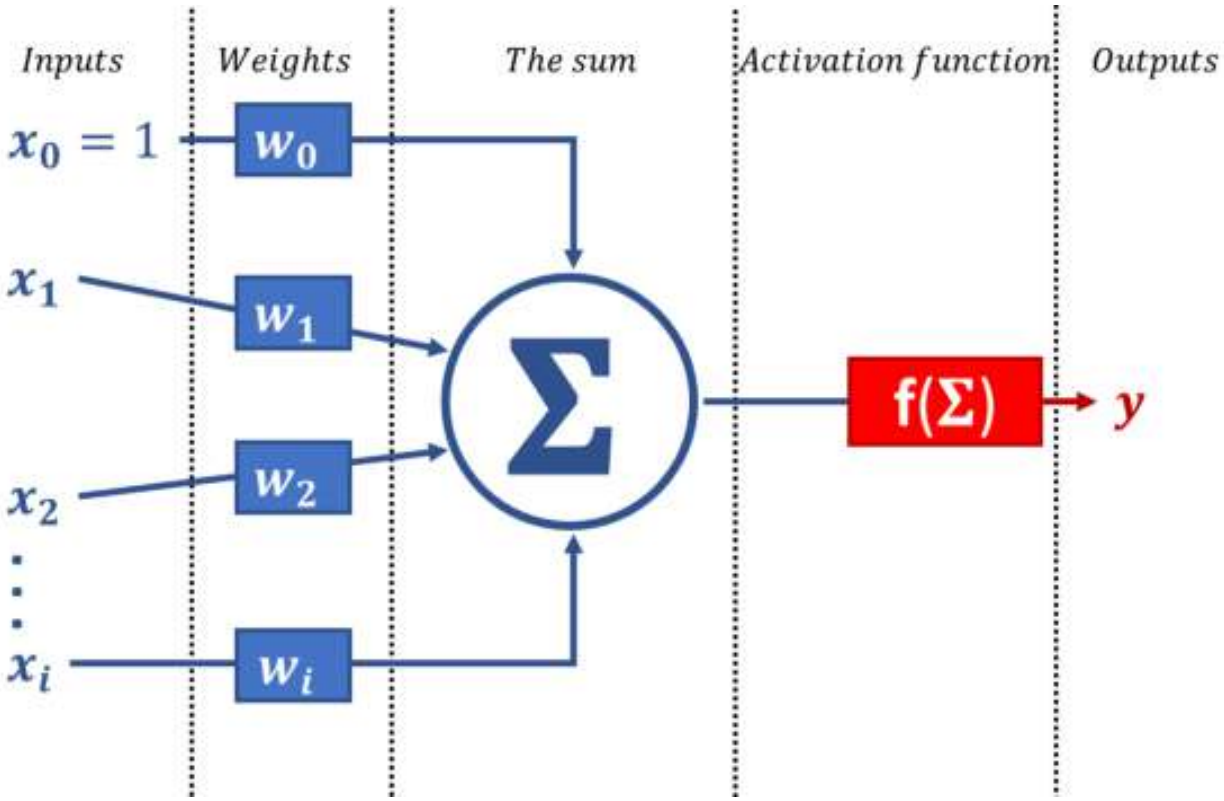

Рис. 1.2

(*https://miro.medium.com/max/625/1*4eB31G5DMgPz-zB0jGk_iA.png)*

У 1959 році Бернард Відроу та Марсіан Хофф розробили моделі нейронних мереж під назвою ADALINE та MADALINE. Це були перші нейронні мережі, які мали прикладне застосування для вирішення реальних проблем. [5]

На даний момент нейронні мережі мають застосування у надзвичайно великій кількості областей. Розпізнавання обличчя, автоматизація, передбачення подій, роботизація, self-driving автомобілі тощо.

Невеликий перелік областей, у яких можна застосувати нейронні мережі:

- Класифікація
- Передбачення (майбутнього, на основі минулих подій)
- Прогнозування
- Розпізнавання
- Кластеризація

1.3 Будова штучних нейронних мереж

Одиниця обчислення в штучній нейронній мережі - це нейрон, так само як і в живому організмі. Його задача - отримати значення, обробити його та повернути результат.

Іншими словами, штучний нейрон - це математична функція. Вона приймає на вхід набір чисел та повертає на вихід одне число.

Результат функції може бути як результатом всієї нейронної мережі, так і вхідними даними для наступних нейронів [6].

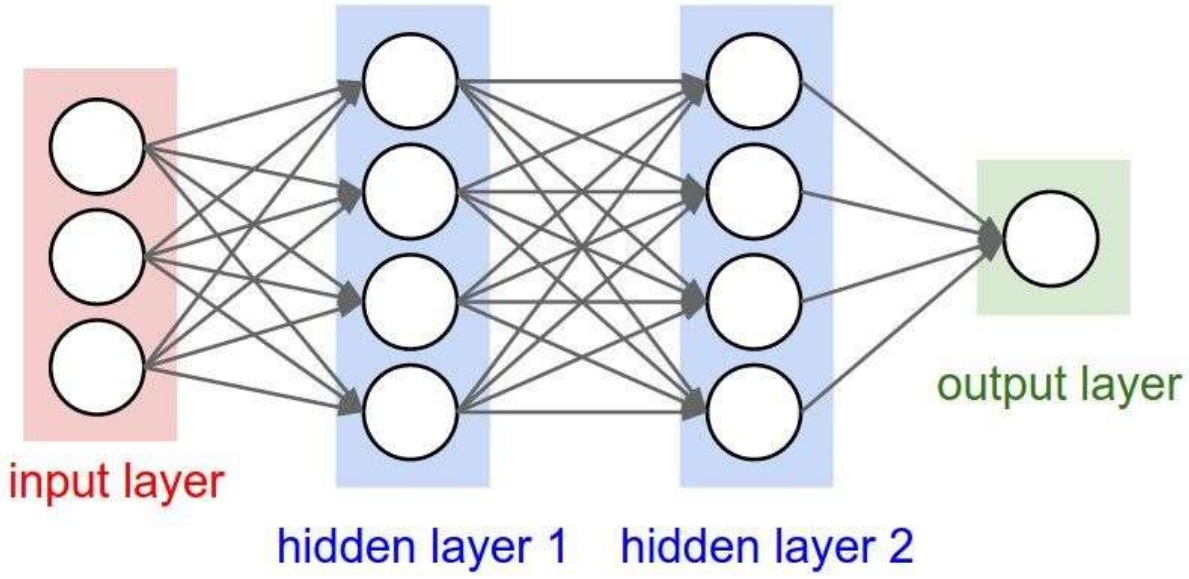

Рис. 1.3

(*https://i.pinimg.com/originals/2d/a1/20/2da120014faf76c47fa4294c7206e291.jpg)*

На рисунку 1.3 - приклад простої нейронної мережі. Вона складається із трьох слоїв(layers).

1. Input layer
2. Hidden layer(s)
3. Output layer

Output кожного слоя - це також input для наступного слоя. Input layer - це вхідні дані. Такий слой може бути тільки один. Після нього - нуль чи більше hidden layers. Це - нейрони, задача яких обробити вхідні дані та виконати основну роботу. Останнім іде output layer - це результат обробки вхідних даних.

Таким чином, на виході кожен нейрон повертає одне число, а кожен слой повертає набір чисел (вектор). Розмірність цього вектору дорівнює кількості нейронів у слої.

1.3.1 Що робить кожен окремий нейрон?

Після того, як нейрон отримав вхідні дані, перед поверненням результату робиться три операції[7]:

1. Вхідний вектор множиться на вектор ваг (weights vector)
2. До отриманого числа додається число bias (“зміщення”)
3. До отриманого числа застосовується функція активації (activation function)

Таким чином, значення нейрона X = activation(w * i + b), де w - вектор ваг, i - вхідний вектор, b - bias, X - результат (число), activation - функція активації.

Якщо розмірність Input Layer - (1, 3), як на Рис 1.3, то розмірність вектору ваг кожного нейрону в hidden layer 1 буде (3, 1).

Функція активації (*activation function, transfer function)* як правило використовується нелінійна.

1.3.2 Функції активації

Розглянемо деякі з найбільш популярних функцій активацій:

ReLU (Rectified Linear Unit) - це функція вигляду max(input, 0). Іншими словами, для позитивних вхідних значень, функція повертає це значення. Якщо вхідні дані < 0, функція повертає 0. [8]

Ця функція була натхненна біологічними нейронами, які можуть або “відпрацювати” або ні. У поєднанні з bias, який міститься в нейроні, ReLU може фільтрувати, щоб значення не виходили за певні межі.

Наприклад, якщо значення bias “-x” - будь яке значення, яке менше x, після додавання bias буде менше 0, і відповідно значення ReLU буде 0.

Графік показано на Рис 1.4

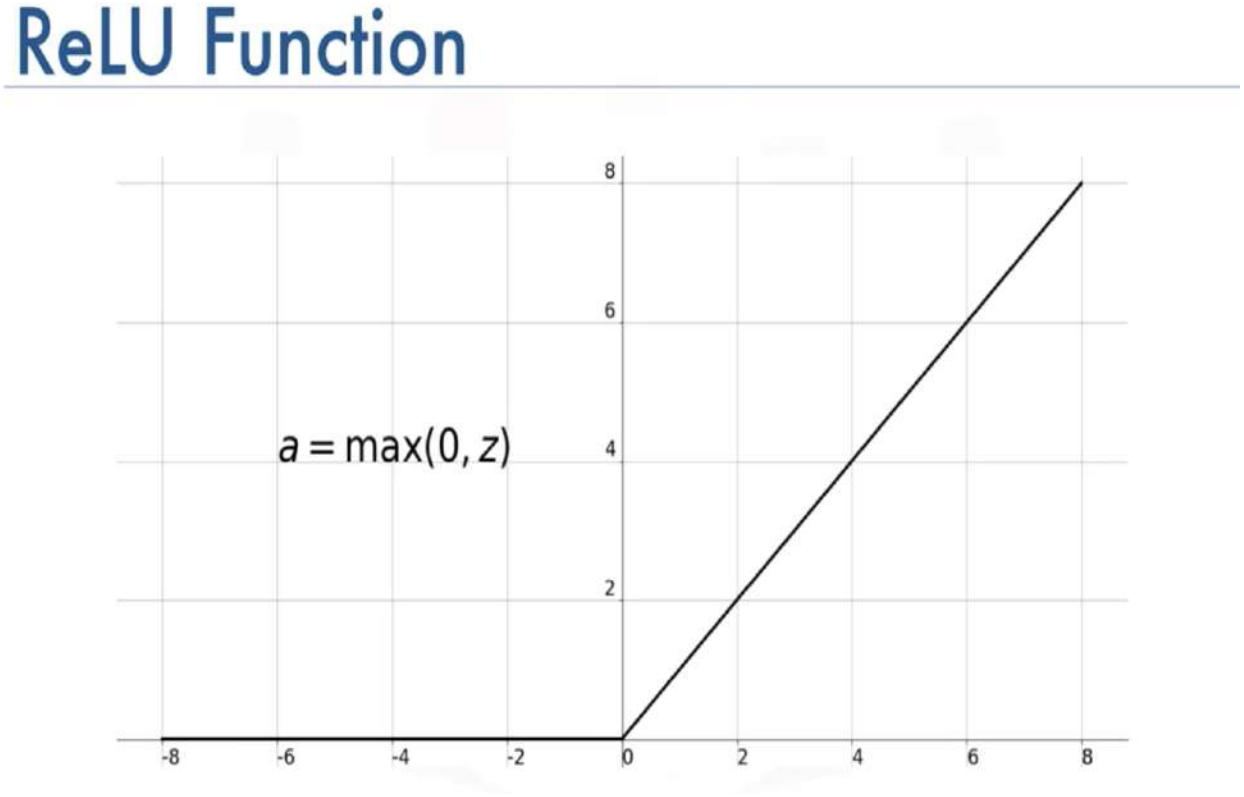

Рис. 1.4

(*https://miro.medium.com/max/3228/1*LiBZo_FcnKWqoU7M3GRKbA.png)*

Ще одна функція активації - **Sigmoid**. Вона отримує на вхід число і повертає значення від 0 до 1, використовуючи функцію на Рис 1.5

$$f(x) = \frac{1}{1 + e^{-x}}$$

Рис. 1.5

(скріншот із

https://towardsdatascience.com/why-do-neural-networks-need-an-activation-function-3a5f6a5f00a)

Дана функція є симетричною, і повертає ½, якщо вхідне число - 0. Графік функції Sigmoid показано на Рис 1.6

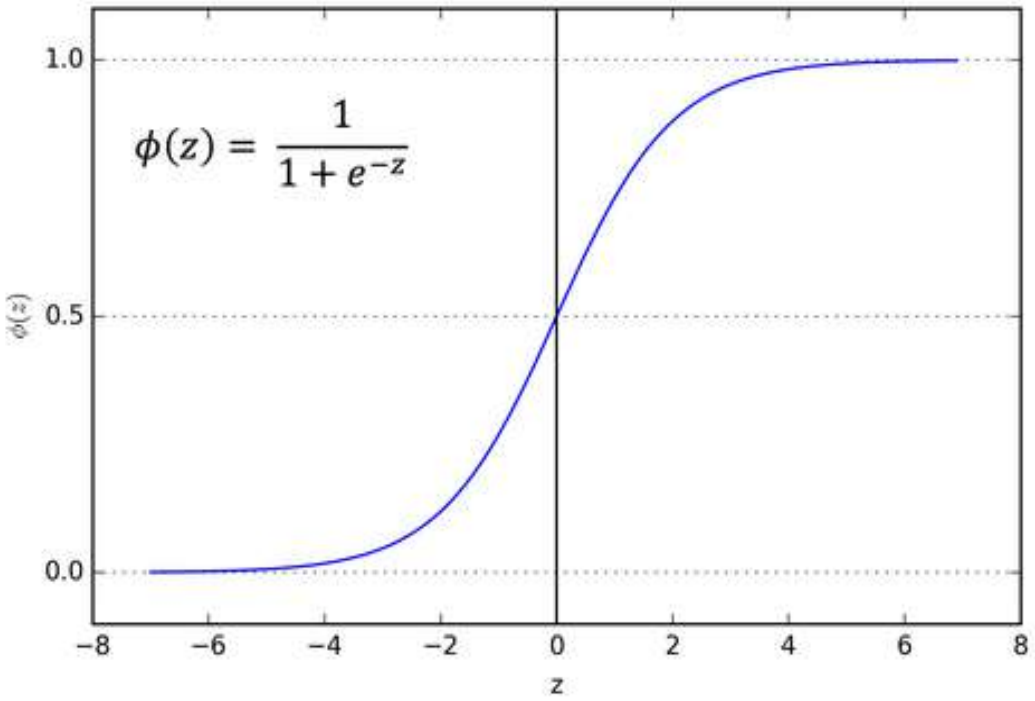

Рис. 1.6

(*https://miro.medium.com/max/970/1*Xu7B5y9gp0iL5ooBj7LtWw.png)*

Ще одним прикладом функцію активації є **TanH**. Її графік схожий на графік Sigmoid, але функція може набувати значень від -1 до 1. Обраховується за формулою на Рис 1.7

$$y(x) = \frac{sinhx}{coshx} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Рис. 1.7

(*https://miro.medium.com/max/249/1*m4n4NXE41sU1CLASyKAZwg.png)*

Графік функції TanH показано на Рис 1.8

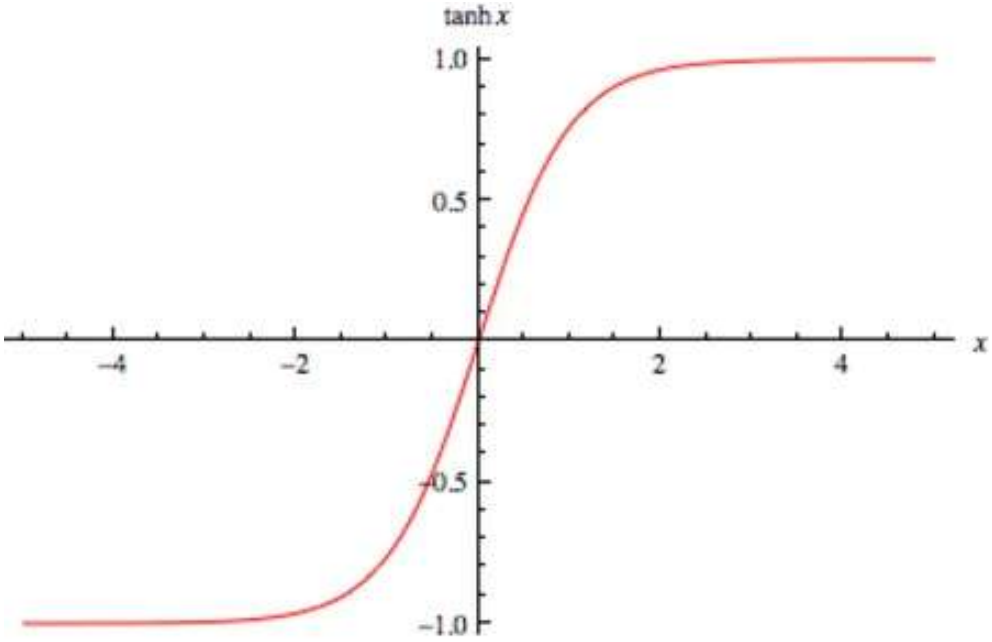

Рис. 1.8

(*https://i0.wp.com/sefiks.com/wp-content/uploads/2017/01/tanh.png?resize=456%2C300&ssl=1)*

1.4 Навчання штучних нейронних мереж

Навчання (тренування) нейронних мереж ділиться на два типи:

- Навчання з вчителем
- Навчання без вчителя

1.4.1 Навчання з вчителем

Навчання з вчителем (англ. *Supervised learning*) - процес навчання нейронної мережі, де використовуються дані які мають “відмітку” (label). Тобто в залежності від задачі, у даних вже є якась “позначка”, яка показує правильну відповідь (до якого класу відноситься запис, чи запис коректний чи некоректний тощо). [10]

Якщо вхідні дані ніяк не марковані - навчання з вчителем застосувати в такій ситуації неможливо.

Оскільки нейронна мережа, яка тренується, заздалегідь знає правильні відповіді, основна її мета - підібрати такі значення в нейронах, щоб похибка була мінімальною.

Вихідними даними такої нейронної мережі може бути класифікація чи регресія.

Класифікація - визначення, до якої групи належать вхідні дані. Наприклад, розпізнавання літер по картинці або ж розділення картинок з бананами від картинок з апельсинами. Визначення, чи є лист спамом - також одне із застосувань для нейронних мереж з класифікацією.

Регресія - це отримання числового значення, використовуючи вхідні дані. Наприклад, можна визначити скільки має коштувати нерухомість, базуючись на даних з ринку. Також можна спробувати передбачити ціну акцій в майбутньому, спираючись на історичні дані.

Приклад, як візуально регресія відрізняється від класифікації зображено на Рис 1.9

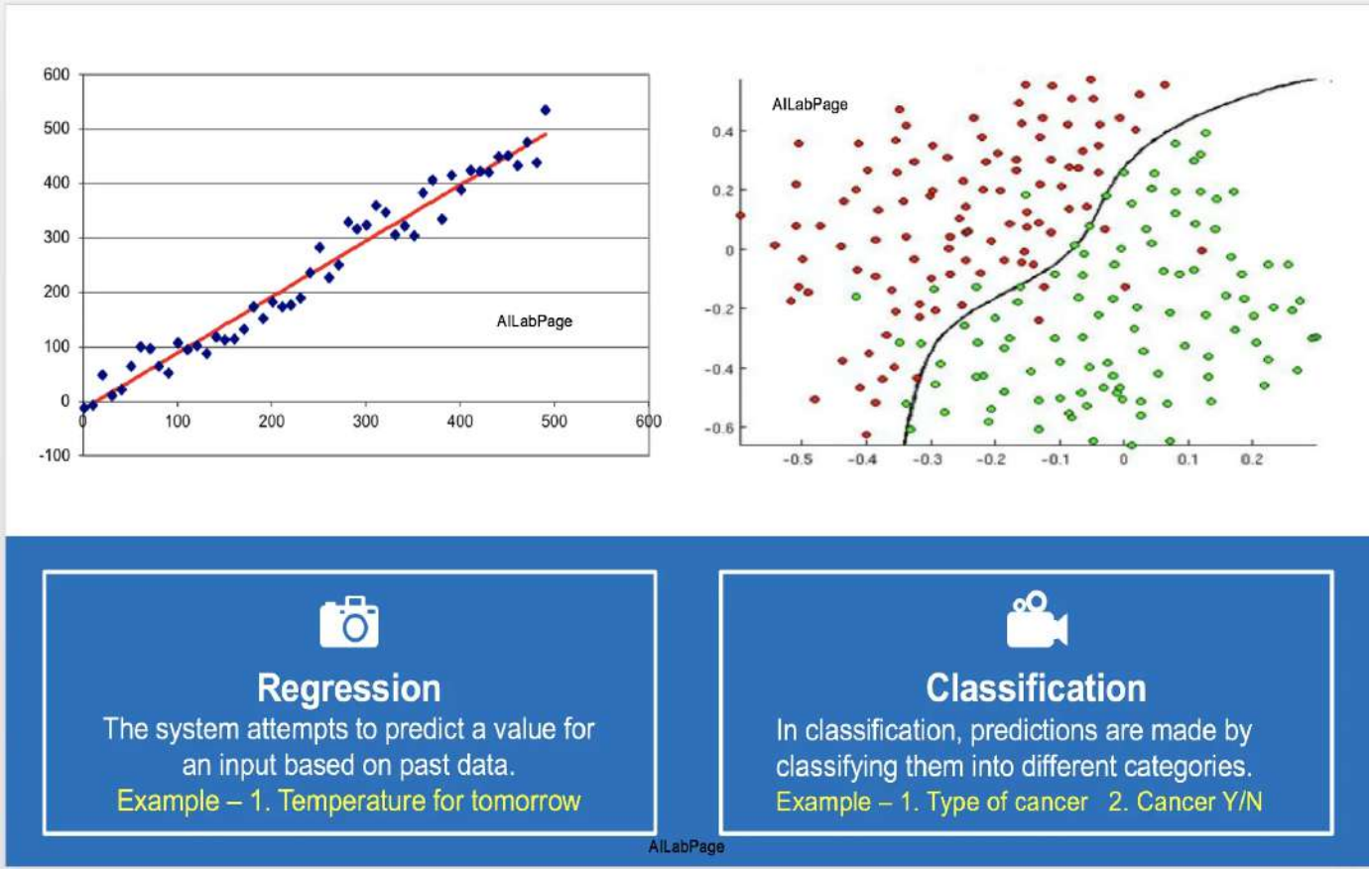

Рис. 1.9

(*https://i0.wp.com/sefiks.com/wp-content/uploads/2017/01/tanh.png?resize=456%2C300&ssl=1)*

1.4.2 Навчання без вчителя

Навчання без вчителя не використовує дані з “відмітками” (*labeled data).* У цьому випадку нейронна мережа сама пробує знайти якісь патерни та дізнатися нову інформацію.

Мінусом є те, що результати в такому випадку важче передбачити, ніж у навчанні з вчителем.

Проте великою перевагою навчання без вчителя є те, що не треба готувати велику кількість тестових даних, необхідних для тренування мережі.

Нейронні мережі без вчителя часто використовуються в таких випадках:

- Кластеризація
- Пошук асоціацій (дозволяє знаходити зв’язки та якісь закономірності між існуючим даними)
- Знаходження аномалій в даних

На Рис 1.10 показано приклад для інтуїтивного розуміння, що таке кластеризація.

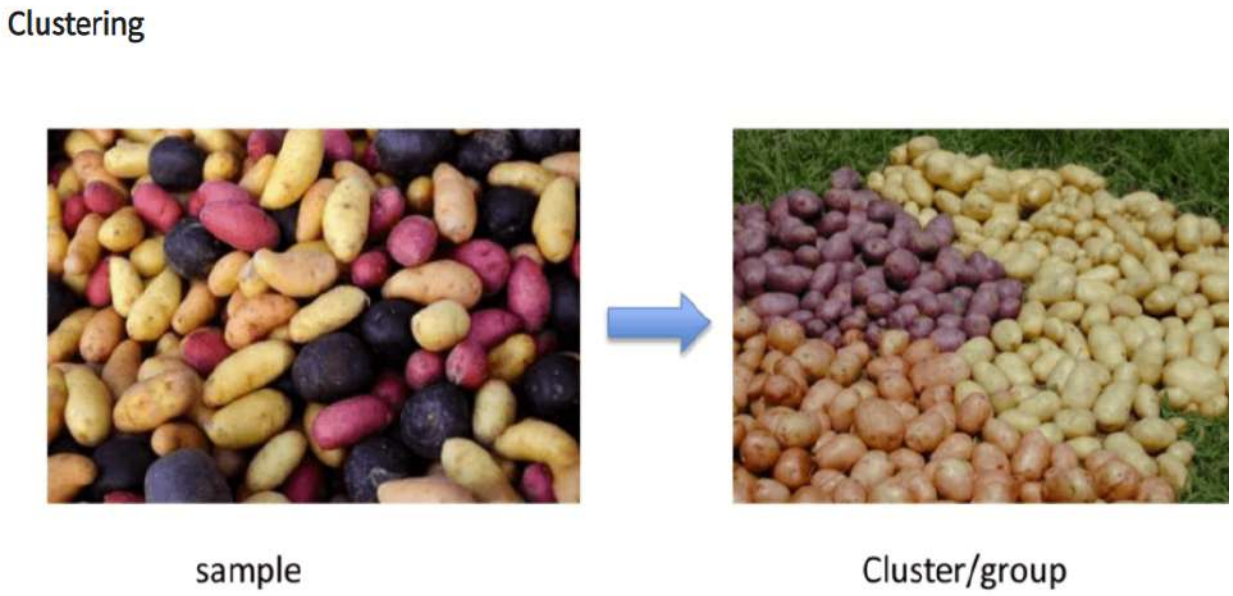

Рис. 1.10

(скріншот з *https://www.guru99.com/supervised-vs-unsupervised-learning.html#2)*

1.5. Типи нейронних мереж

1.5.1 Feed-forward neural networks

Feed-forward neural networks - зображено на Рис 1.11. Цей тип нейронної мережі був винайденим один із перших і є найпростішим. Свою назву він

отримав, оскільки дані йдуть в одному напрямку, від input до output. Немає циклів, на відміну від RNN.[11]

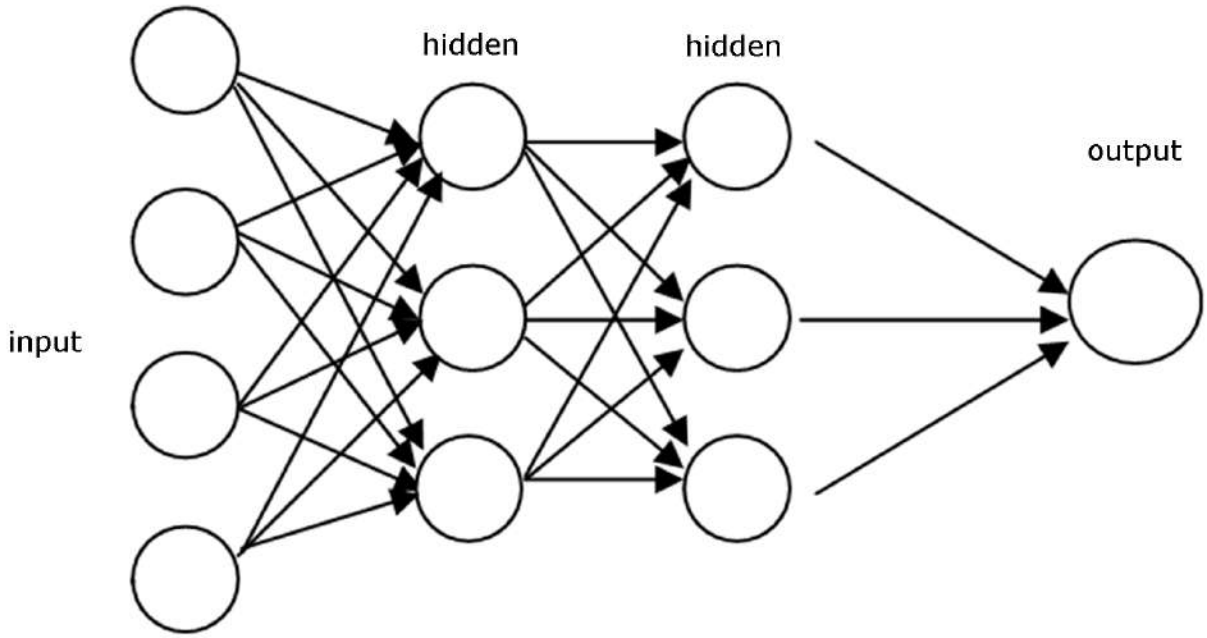

Рис. 1.11

(https://static.packt-cdn.com/products/9781788397872/graphics/1ebc2a0a-2123-4351-b7e1-eb57f098bafa.png)

Дана нейронна мережа використовується переважно в навчанні з вчителем. У такій мережі кожен нейрон із одного слоя пов’язаний з кожним нейроном із наступного слоя.

Перевагою такої мережі є те, що вона “не знає” про структуру вхідних даних, тобто вона не є створеною спеціально для обробки зображень чи тексту. Але недоліком є те, що її продуктивність на практиці нижча за вузьконаправлені нейронні мережі.[12]

1.5.2 Recurrent neural networks

Recurrent neural networks - цей тип нейронних мереж, який добре працює з “послідовними” (*sequence*) даними. Наприклад з текстом, аудіо чи відео, даними про погоду тощо.

На відміну від feed-forward нейронних мереж, де дані проходять в одну сторону, і кожен слой відпрацьовує тільки один раз, у RNN це не так. У RNN є пам’ять. І якщо feed-forward мережа буде обробляти, наприклад, речення - це буде відбуватися по одному слову, тобто вона не буде враховувати структуру речення. У RNN це враховується.[13]

Схематично порівняння feed-forward нейронних мереж і RNN показано на Рис 1.12

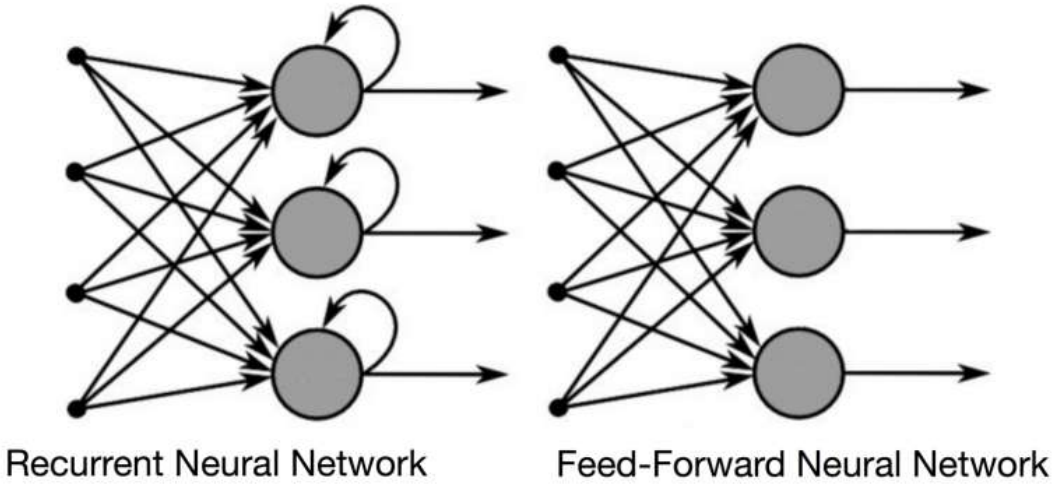

Рис. 1.12

(https://builtin.com/sites/default/files/styles/ckeditor_optimize/public/inline-images/rnn-vs-fnn.png)

Оскільки в RNN є інформація про минуле, то ваги(weights) в нейронах є не тільки для теперішніх вхідних даних, а й для попередніх.

У той час як у feed-forward мереж зв’язок вхідних та вихідних даних один до одного, у RNN це може бути один до одного, один до багатьох, багато до одного то багато до багатьох, як показано на Рис 1.13

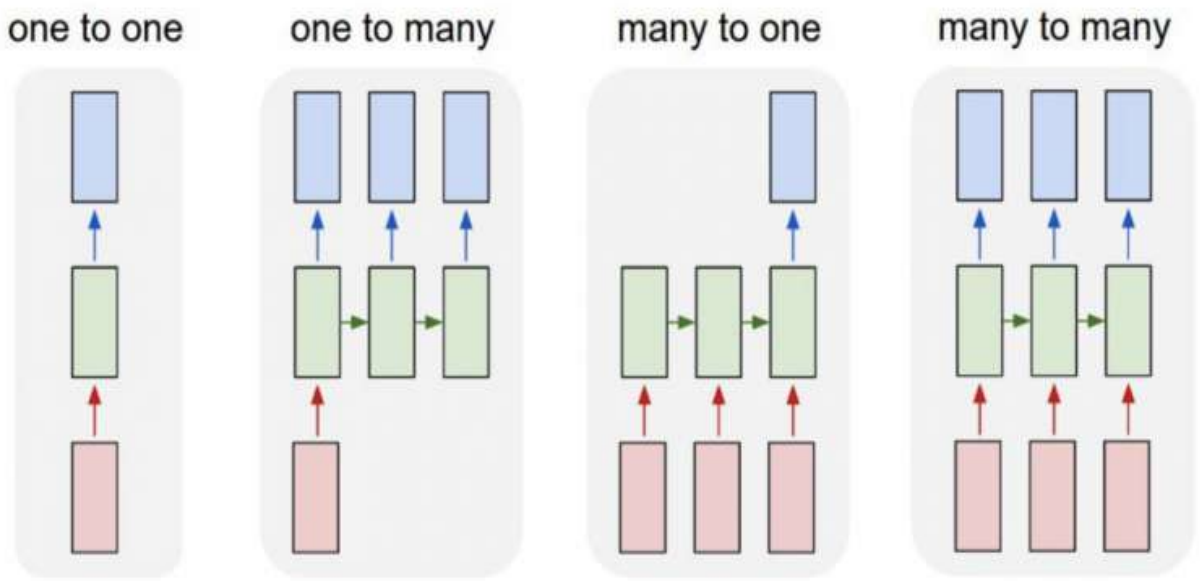

Рис. 1.13

(https://builtin.com/sites/default/files/styles/ckeditor_optimize/public/inline-images/Feed-Forward-Neural-Networks.png)

Часто рекурентні нейронні мережі застосовують для перекладу текстів, прогнозування слів у тексті (наприклад як це робить Google у пошуковій стрічці). Також RNN може застосовуватись для формування тексту, використовуючи відеозапис, як це показано на Рис 1.14

Рис. 1.14

(https://tektoks.files.wordpress.com/2018/08/visionresult1.jpg)

1.5.3 Convolutional Neural Networks

Convolutional Neural Networks (також *CNN* або *ConvNet***)** – це нейронні мережі, які вміють добре працювати з зображеннями.

Перша CNN називалася LeNet і була придумана в 1990х роках. Типова архітектура CNN має три типи слоїв: convolutional layer, pooling layer та fully connected layer.[14] Це схематично показано на Рис 1.15

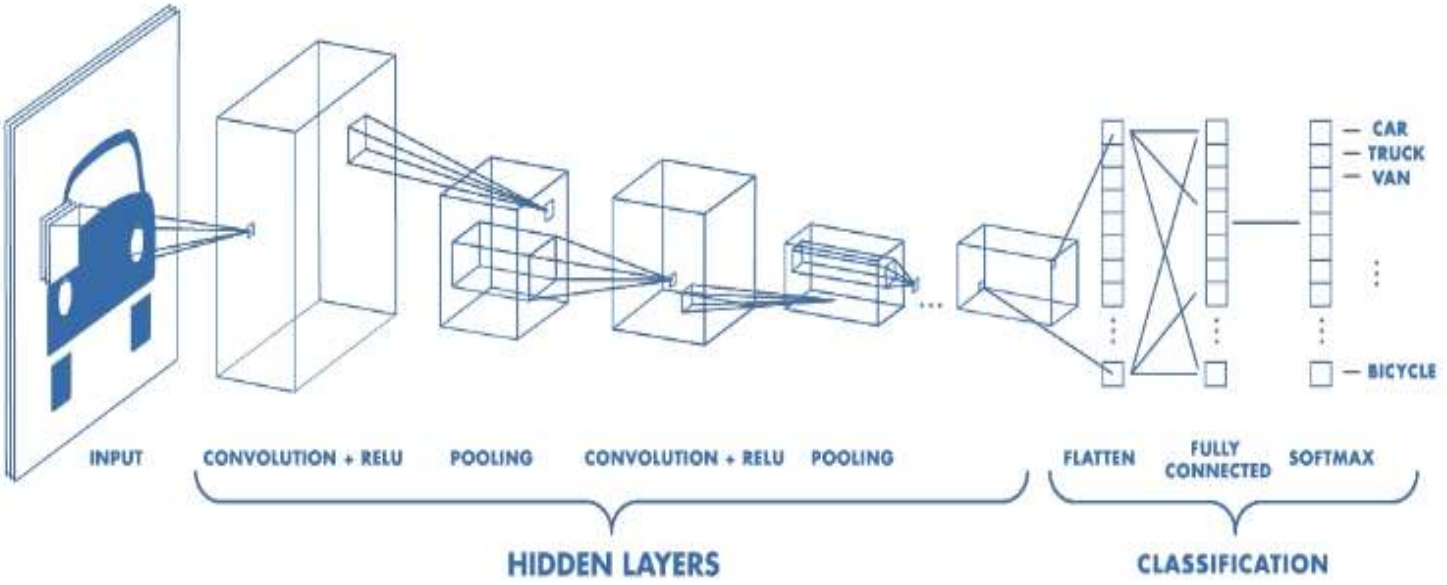

Рис. 1.15

(https://tektoks.files.wordpress.com/2018/08/visionresult1.jpg)

Convolutional layer є основним “будівельним матеріалом” в CNN. Ідея в тому, що ці слої вміють знаходити певні риси (features) зображення, наприклад кути, лінії, якісь кольорові гами тощо. На перших слоях це можуть бути дуже прості патерни, але з кожним наступним слоєм знайдені патерні можуть ускладнюватися.

Це робиться завдяки так званим “фільтрам” (англ. *kernel*). Фільтри вміють розпізнавати якісь патерни в зображеннях. Схематично це показано не рисунку 1.16. Замість літер у реальному світі будуть числа, які, наприклад, показують RGB значення або значення від 0 до X якщо зображення в форматі grayscale.

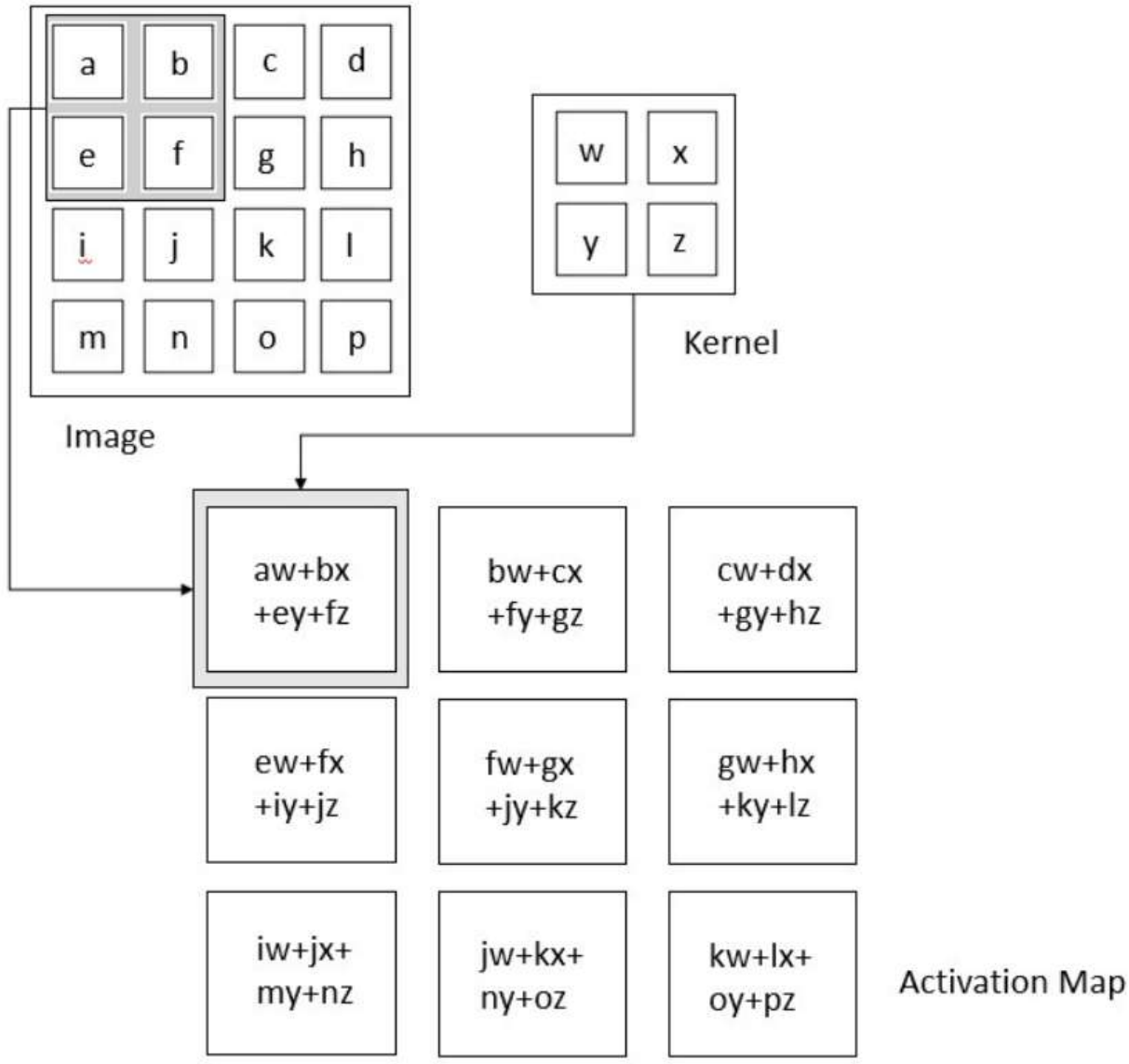

Рис. 1.16

*(https://miro.medium.com/max/875/1*r13ZUdVTQwVuhDPmo3JKag.png)*

Оскільки процес, описаний вище - лінійний, щоб зробити його нелінійним часто додають функцію активації відразу після convolutional слоя. Наприклад Sigmoid, TanH, чи ReLU.

Pooling слої - це слої, які отримують на вхід матрицю, і використовуючи певні правила зменшують її розмір. Це допомагає зменшити кількість параметрів (weights), які мають бути натреновані, тим самим зменшуючи

навантаження на обчислювальний пристрій. Також pooling layer’и дозволяють зменшити так зване “перетренування” (англ. *overfitting)* нейронної мережі.

Є maximum pooling, який обирає найбільше значення з підматриці, як показано на Рис 1.17

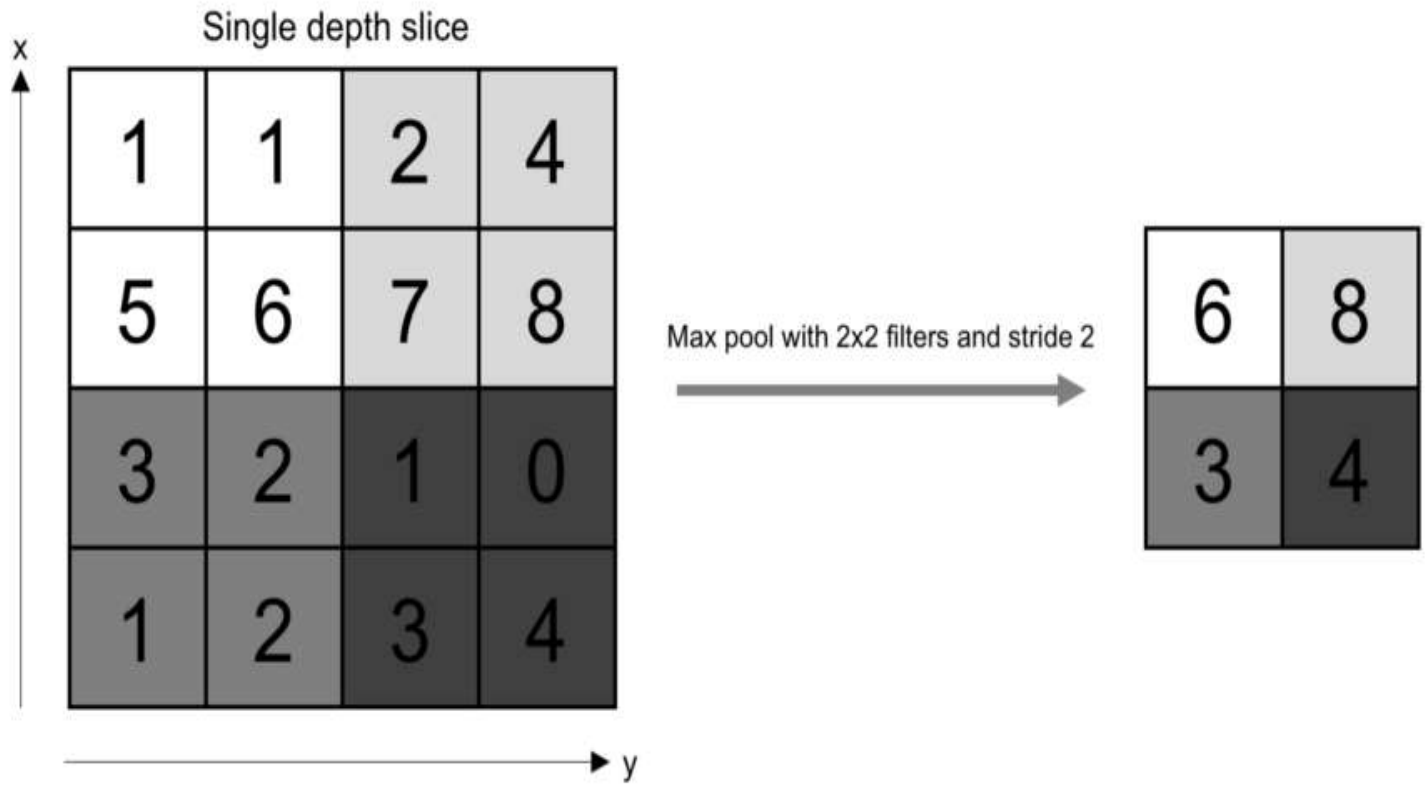

Рис. 1.17

*(https://miro.medium.com/max/875/1*sK7oP1m129V_oNGSsHIm_w.png)*

Також існує average pooling, який обирає середнє значення для кожної підматриці. Приклад показано на Рис 1.18

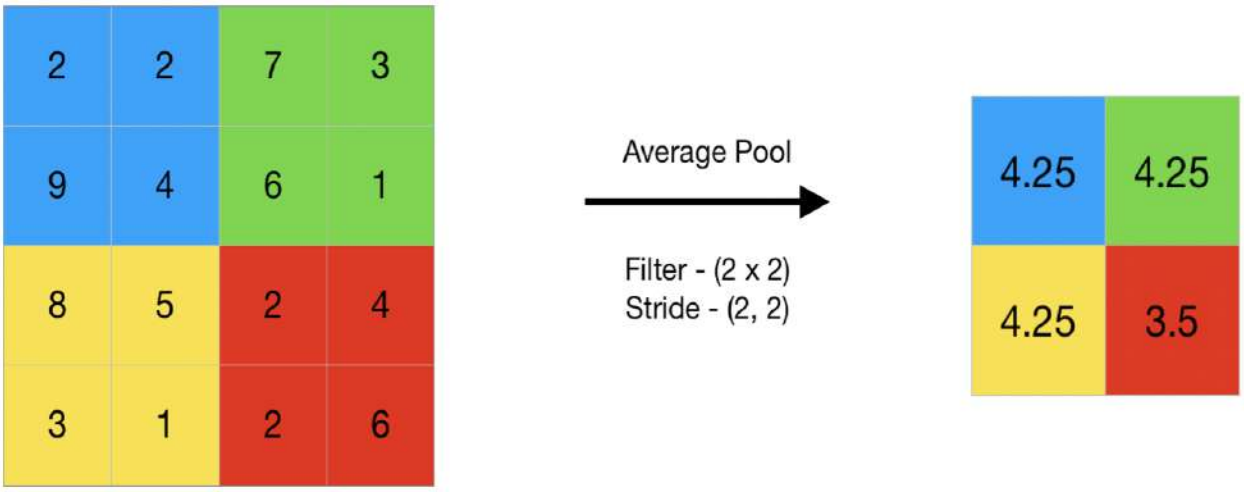

Рис. 1.18

(https://media.geeksforgeeks.org/wp-content/uploads/20190721030705/Screenshot-2019-07-21-at-3.05.56-AM.png)

У **Fully connected** (FC) layers кожен нейрон з одного слоя пов'язаний з кожним нейроном в попередньому та наступному слоях. Точно так само, як це відбувалося в feed-forward нейронних мережах.

Таким чином, convolutional layers відповідають за знаходження певних патернів у зображеннях, а FC слої відповідають за їхню класифікацію.

На Рис 1.19 показано архітектуру CNN, яка називається AlexNet. Два передостанні слої - Fully Connected.

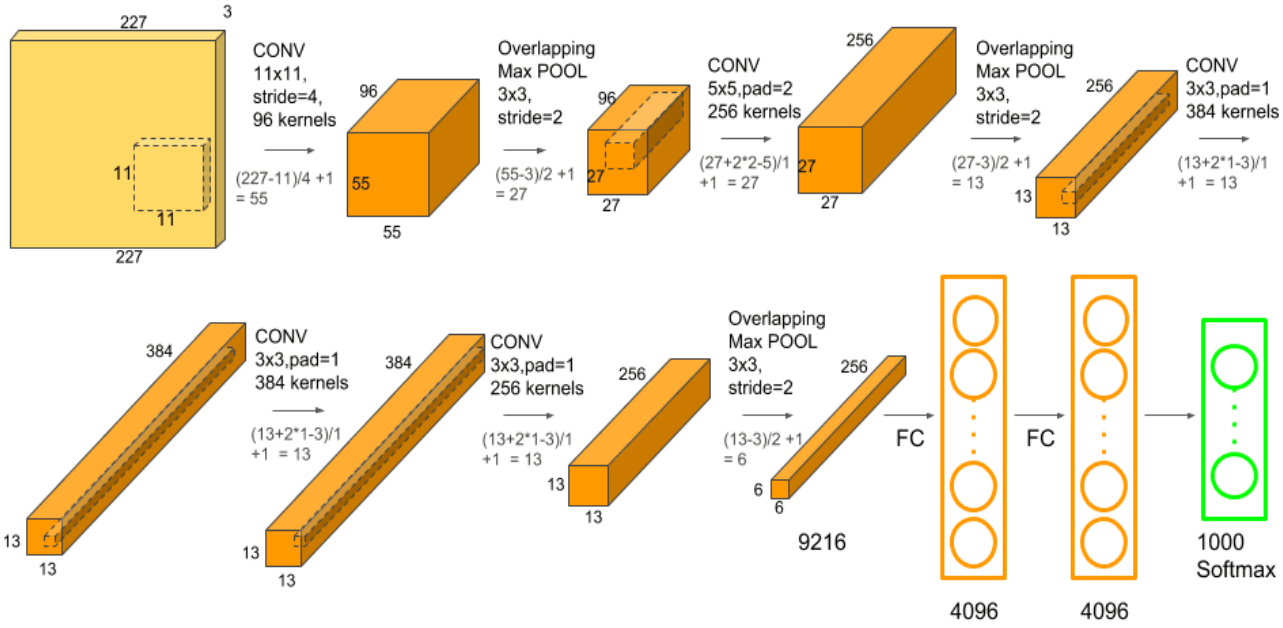

Рис. 1.19

(https://media.geeksforgeeks.org/wp-content/uploads/20190721030705/Screenshot-2019-07-21-at-3.05.56-AM.png)

Розділ 2: Розпізнавання обличчя

2.1 Face detection vs face recognition vs face verification

Термін “розпізнавання” обличчя можна зрозуміти по різному. В англійській мові є слова, які доволі точно описують можливі його значення - detection, recognition та verification.

2.1.1 Face detection

Face detection - це “знаходження” обличчя на фото. Тобто єдина задача такої нейронної мережі - отримати на вхід зображення та повернути 0 чи 1 якщо на ньому є обличчя, або ж межі обличчя (координати) на фото, як показано на Рис 2.1

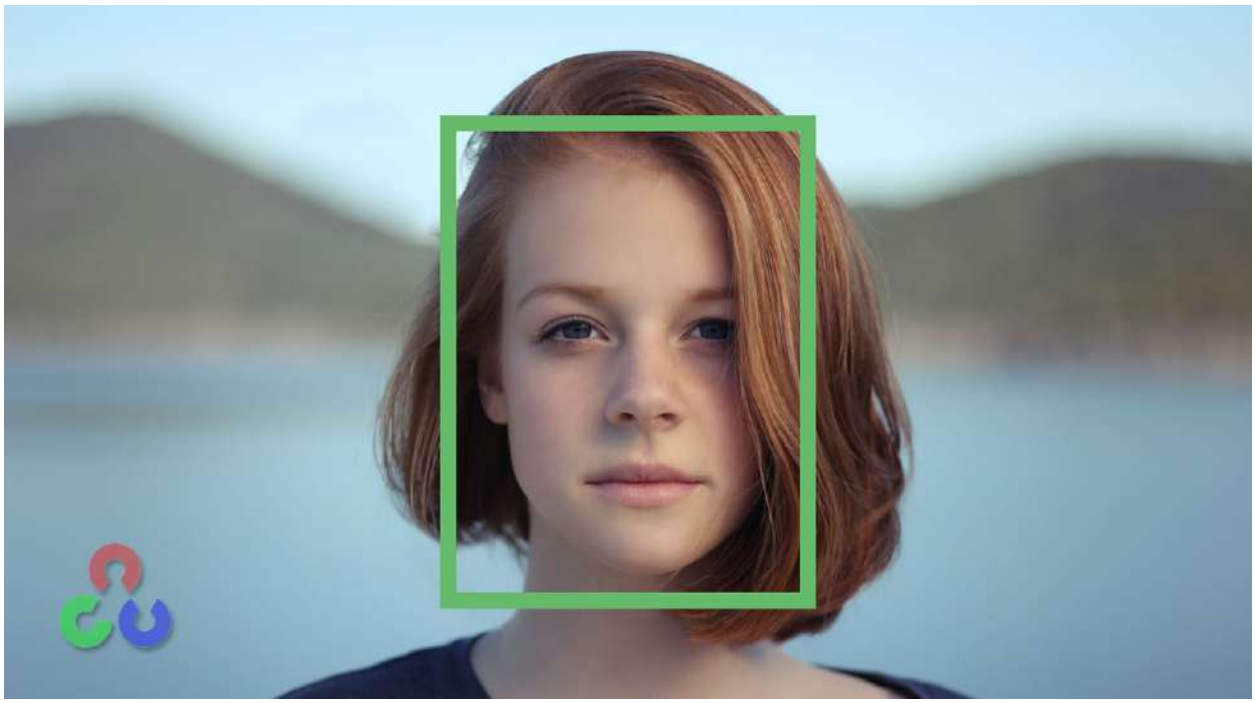

Рис. 2.1

(https://i.morioh.com/2020/02/26/2954199cef43.jpg)

2.1.2 Face verification

Face verification (верифікація) - це щось схоже на аутентифікацію. Наприклад, FaceID, який є в iPhone - це приклад верифікації за допомогою обличчя. На вхід отримується зображення обличчя та ім’я або ідентифікатор персони, і задача нейронної мережі - переконатися, що зображення належить персоні з заданим ідентифікатором. Рішення із face verification також часто перед цим роблять face detection.

2.1.3 Face recognition

Face recognition- це більш широке рішення. Є база даних із K людей. Отримуючи на вхід зображення, нейронна мережа має відповісти, хто із K людей є на даному зображенні (або взагалі ніхто із них). Перед тим як виконувати face recognition часто може виконуватися алгоритм face detection. Далі у дипломній роботі при згадуванні слова “розпізнавання” буде матися на увазі саме face recognition.

Розпізнавання є важчою проблемною, ніж верифікація. Припустимо є верифікаційна система, точність якої 99%. Це може працювати доволі непогано. Але якщо розглянути розпізнавання, за умови що в базі даних для розпізнавання є наприклад 100 людей, а її - точність 99%, то шанс зробити помилку доволі великий.

2.2 One Shot Learning

Одна із проблем, яка виникає при розробці систем, які розпізнають обличчя - це необхідність розпізнавати людину, маючи лише один приклад її

зображення. Така проблема відома як One shot learning problem. А нейронні мережі працюють таким чином, що для тренування їм треба багато прикладів.

Припустимо, ми маємо систему, яка розпізнає співробітників. У нашій системі є чотири співробітника - A, B, C, D. Потім хтось з’являється в офісі і хоче пройти через нашу систему. Назовемо цю людину Input. Схематично все зображено на Рис 2.2

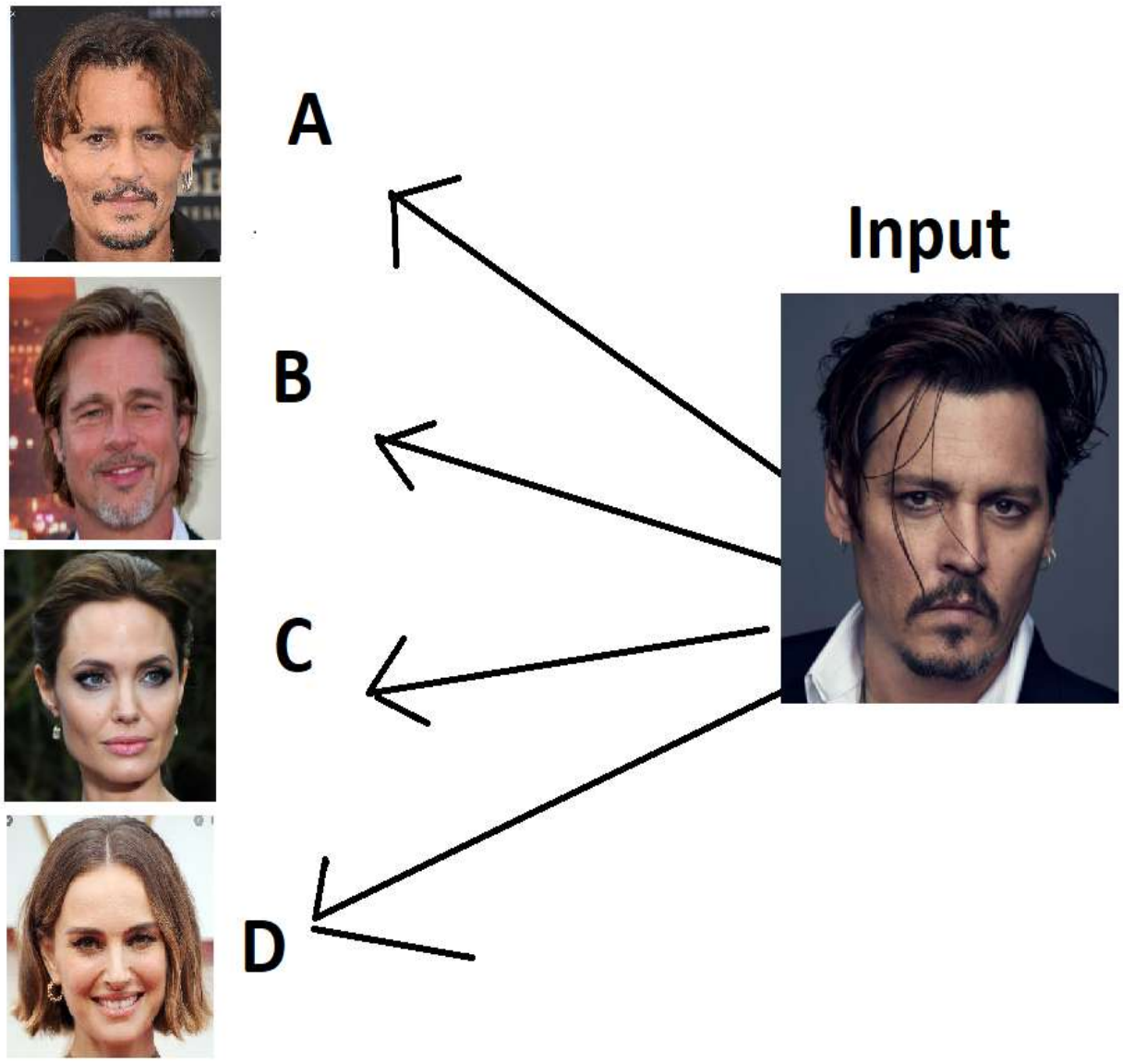

Рис. 2.2

Цілком можливо, що зображень людей A, B, C, D є всього по одному екземпляру в базі. Не дивлячись на це, система має розпізнати, що Input - це співробітник “A”. Сценарій, коли треба розпізнати людину, маючи всього один приклад її фотографії - називається One-shot learning. У реальному світі таке часто буває, оскільки далеко не завжди можна знайти багато фотографій однієї людини.

Першим рішення задачі буде взяти Input зображення, передати його в CNN, і використовуючи softmax(5) - отримати ймовірність співпадіння вхідного зображення з кожним із співробітників, або п’ятий варіант - відсутність співпадіння з будь ким.

Насправді це не спрацює, оскільки для тренування нейронної мережі було занадто мало тестових даних. Також виникає проблема, якщо з’являється нова людина - треба буде навчати нейронну мережу знову.

Рішенням даної проблеми є так звана “функція схожості” (англ. *Similarity function*). Схематично вона виглядає так: difference(image1, image2) = X. Де image1 - перше зображення, image2 - друге зображення, difference - функція, яка показує, наскільки зображення відрізняються один від одного, X - числове значення, наскільки зображення різні. Чим більше X - тим більше зображення відрізняються.

Таким чином, якщо image1 та image2 - одна і та ж людина, функція поверне маленьке значення. Якщо це дві різні людини - значення буде більшим. Також треба підібрати параметр K, який буде певною “межою”. Якщо значення функції менше ніж K - це означає що на двох зображеннях одна і та ж людина.

У прикладі, який наведено на Рис 2.2 ми застосуємо цю функцію чотири рази - порівняємо Input з A, B, C та D відповідно. Оскільки Input та A - одна і та ж людина, у цієї пари значення функції буде найменше.

У наступному розділі буде розглянуто, яким чином можна розробити таку функцію.

2.3 Siamese network

Функція “схожості”, про яку було описано в розділі 2.2, отримує на вхід два зображення та повертає числове значення, яке описує схожість зображень.

Рішенням для даної функції може бути “сіамська мережа” (англ. *Siamese network*).

На рис 2.3 показано приклад типової ConvNet - декілька кроків “згортки”, можливо якісь pooling - слої, в кінці маємо два Fully connected слоя.

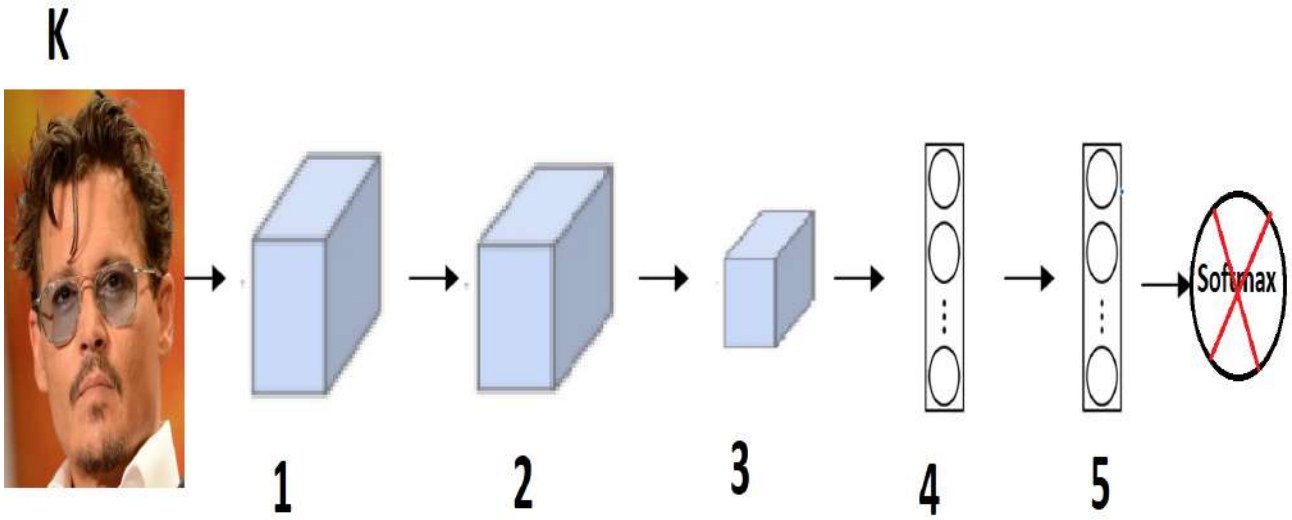

Рис. 2.3

Часто в кінці додають додають функцію активації softmax для класифікації. У нашому випадку така функція не потрібна. Нас цікавить вектор, який формується на п’ятому слої (У реальному житті цей слой може знаходитися далі, числа обрані для спрощеного пояснення).

Припустимо, що довжина цього вектора - 128. Якщо вхідне зображення - K, то вектор, сформований на п’ятому слої можна позначати як f(K), який має довжину 128. Також цей вектор можна назвати “кодуванням” (англ. *encoding або embedding*) вхідного зображення.

Щоб побудувати систему, яка порівнює два обличчя K та K1, треба передати на вхід зображення K та K1 одній і тій самій нейронній мережі з однаковими параметрами. Таким чином у нас буде кодування для кожної картинки - f(K) та f(K1). Приклад показано на Рис 2.4

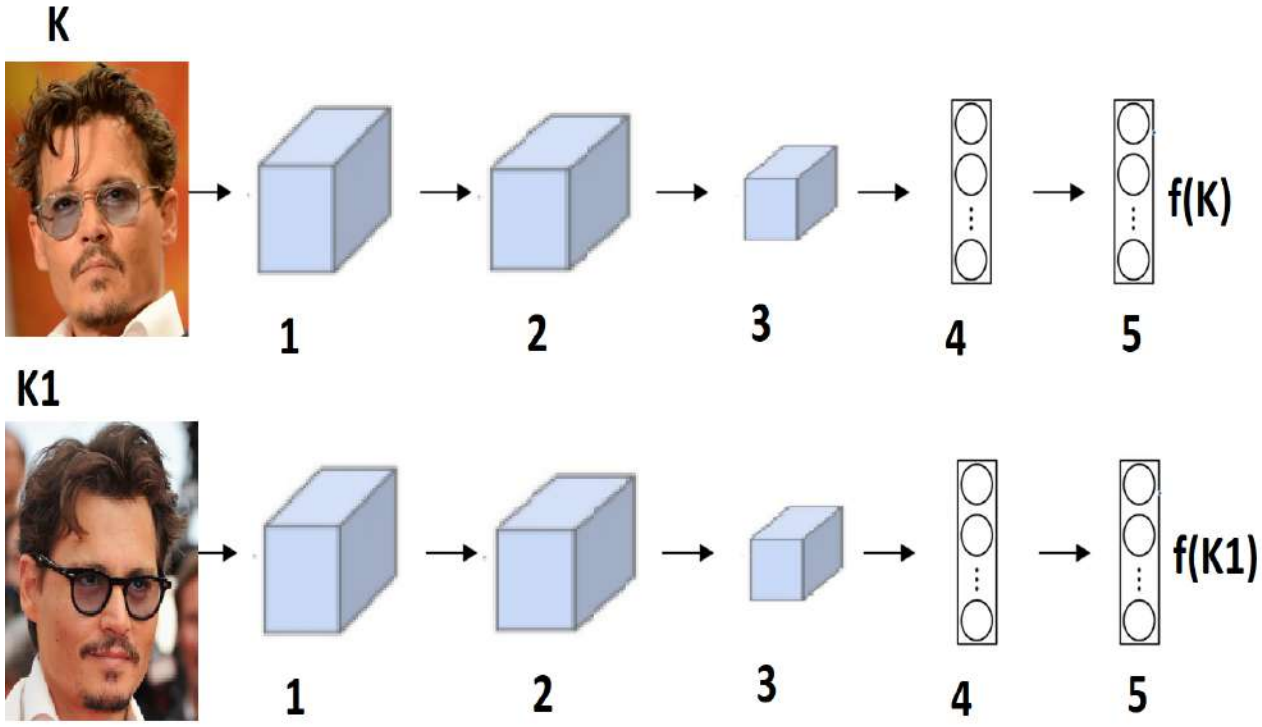

Рис. 2.4

Тепер ми можемо визначити функцію дистанції (d) між K та K1. Це буде дистанція між векторами K та K1 відповідно. Чим дистанція більша, тим сильніше зображення відрізняються один від одного

$$d(K, K1) = || f(K) - f(K1) ||^2$$

Тепер наша мета - підібрати параметри в нейронній мережі таким чином, щоб справджувалися ці два правила:

1. Якщо K та K1 - одна і та ж людина, то функція d(K,K1) має повертати низьке значення
2. Якщо K та K1 - різні люди, то функція d(K,K1) має повертати високе значення

Є декілька способів, як підібрати значення для параметрів. Ми розглянемо Triplet loss алгоритм та бінарну класифікацію.

2.4 Triplet loss

Підібрати параметри для нейронної мережі можна використовуючи градієнтний спуск (англ. *Gradient descent*) та Triplet-loss функцію.

Рис 2.5

Розглянемо Рис 2.5. На ньому три зображення - R, P, N. R - це зображення людини, P - інакший приклад цієї ж самої людини, N - зображення іншої людини. Ми хочемо щоб значення d(R,P) було мінімальним, d(R,N) - високим. Алгоритм називається Triplet loss, оскільки в ньому три вхідних параметри - R, P та N.

Можемо виразити це в формулах: Ми хочемо отримати d(R,P) <= d(R,N). Звідси маємо:

$|| f(R) - f(P) ||^2 \leq || f(R) - f(N) ||^2$, де f - "кодування" зображення.

Ми можемо перенести праву частину формули вліво і отримати такий вираз:

$|| f(R) - f(P) ||^2 - || f(R) - f(N) ||^2 \leq 0$

Одним із тривіальних рішень, яке задовольняє нашу формулу - змусити f повертати 0 для будь яких вхідних даних. За таких умов ми маємо завжди буде ми отримувати $0 - 0 \leq 0$

Ще одним тривіальним рішенням буде, щоб функція f завжди повертала один і той самий результат X, незалежно від вхідних даних. Тоді ми матимеро $X - X \leq 0$.

Очевидно, що такі рішення нам не підходять. Щоб уникнути ситуацій, коли нейронна мережа приходить до таких рішень, треба внести зміни у формулу. Ми можемо сказати, що різниця d(R,P) - d(R,N) має бути не просто менше або дорівнювати нулю, а ця різниця має бути трішки меншою за нуль.

За це буде відповідати гіперпараметр α, і тепер формула має вигляд:

$|| f(R) - f(P) ||^2 - || f(R) - f(N) ||^2 \leq 0 - \alpha$

Тепер тривіальні рішення, які ми розглянули, не будуть працювати. Часто гіперпараметр записують зліва для зручності, і тоді формула набуває вигляду:

$$|| f(R) - f(P) ||^2 - || f(R) - f(N) ||^2 + \alpha \leq 0$$

Припустимо ми вибрали α= 0. 3. Тоді якщо d(R,P) = 0.7, а d(R,N) = 0.75, хоча різниця є, але ми не будемо задоволені, оскільки вона менша ніж α.

У даному випадку ми можемо збільшити d(R,N) як мінімум до 1. Також ми можемо зменшити d(R,P) до 0.35 чи менше. Також можемо трішки зменшити d(R,P) і трішки збільшити d(R,N). Головне, щоб різниця між ними була більше за α.

Отже, роль α - "відштовхувати" значення d(R,P) та d(R,N) один від одного. Чим більше значення α - тим більша має бути різниця між цими двома значеннями.

Тепер визначимо Triplet loss функцію. Як було описано, вона приймає на вхід три параметри - R, P, N. R та P - одна і та ж людина, N - інша.

Loss-функція (L) матиме наступний вигляд:

$$L(R, P, N) = max(|| f(R) - f(P) ||^2 - || f(R) - f(N) ||^2, 0)$$

Ми додали функцію max. Таким чином, якщо значення першого аргументу менше ніж нуль, ми завжди отримуємо нуль. Якщо перший аргумент повертає значення більше за нуль - функція max поверне дане значення, отже Loss буде позитивним, тобто нейронну мережу треба покращувати.

Це була Loss-функція для трьох зображень, тепер опишемо Loss-функцію для всієї нейронної мережі:

$$L_{total} = \sum_{i=1}^{m} L(R^{(i)}, P, N^{(i)})$$

Це буде сума всіх Loss-функцій, обрахованих для кожного прикладу із тренувального набору даних для нейронної мережі.

Для тренування можна використати набір із тисячі людей та десяти тисяч зображень цих людей. Зображень має бути більше ніж людей (у наведеному прикладі в середньому кожна людина має 10 зображень), оскільки для того, щоб добре підібрати параметри в нейронній мережі, треба багато даних. Маючи по одному зображенню кожної людини, така нейронна система не зможе навчитися. Але після тренування нейронна мережа буде вирішувати проблему, про яку ми говорили раніше - One shot learning problem.

2.4.1 Як підібрати триплети зображень

Для тренування мережі нам треба триплети (R, P, N) зображень. Якщо такі триплети генеруються випадковим чином із усього набору зображень, то нейронну мережу важче буде навчити, оскільки дуже великий шанс, що R та N будуть дуже сильно відрізнятися.

Треба підібрати для тренування такі триплети R, P ,N, на яких буде “важко” навчитися нейронній мережі. Тобто d(R,P) та d(R,N) мають не сильно відрізнятися, щоб нейронна мережа була максимально ефективною.

Приклад частини тестових даних для тренування нейронної мережі наведено на Рис 2.6

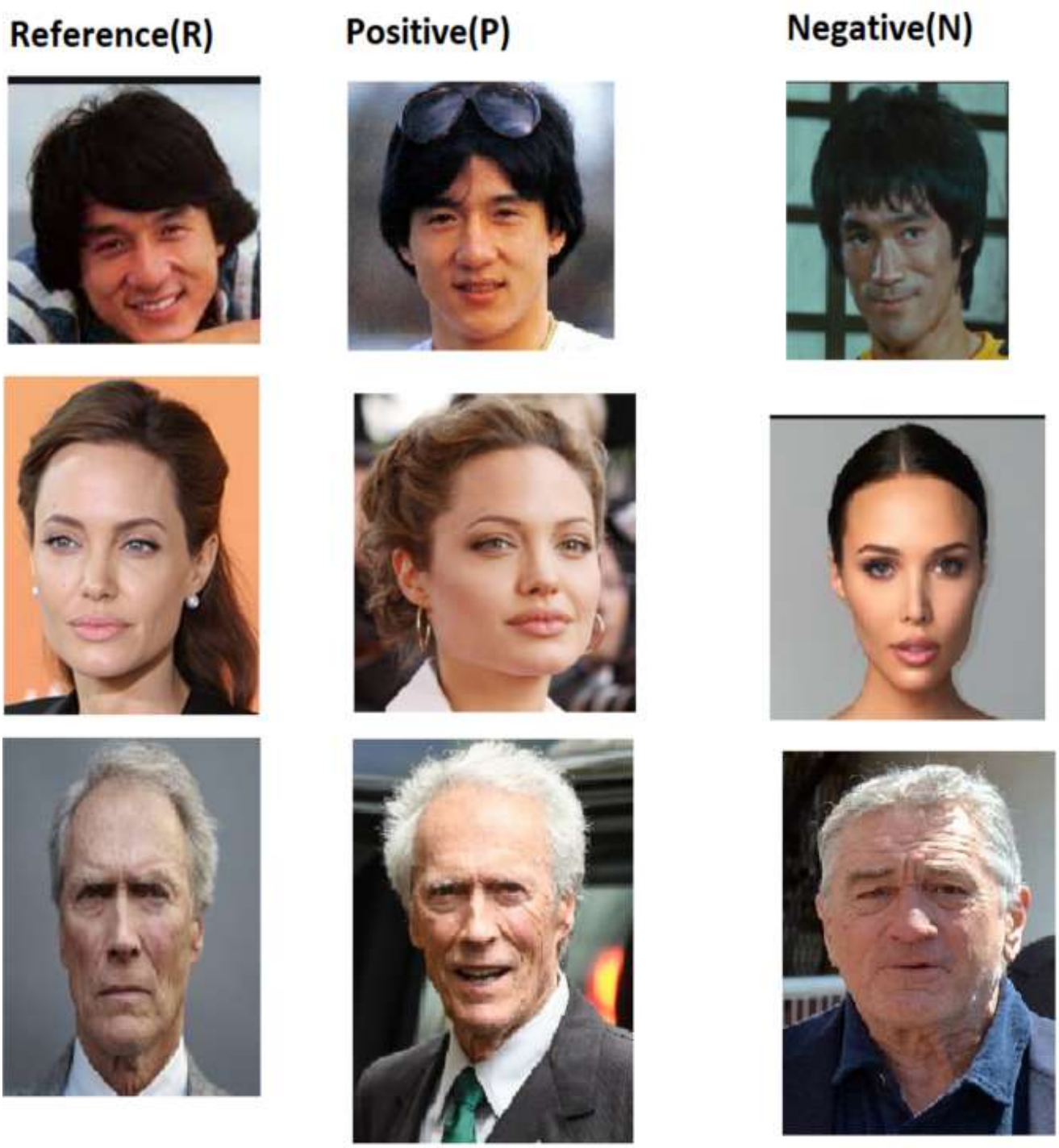

Рис 2.6

Сучасні комерційні системи які, займаються розпізнаванням облич, тренуються на дуже великих масивах даних - десятках та сотнях мільйонів зображень.

Деякі компанії після того, як натренували нейронну мережу на великих масивах даних, викладають у відкритий доступ натреновані параметри для нейронної мережі.

Таким чином будь-хто з бажаючих може користуватися вже натренованою нейронною мережею, не маючи під рукою мільйони зображень та обчислювальних потужностей для тренування мережі.

2.5 Binary classification

Ми розглянули Triplet loss алгоритм для тренування нейронної мережі, яка називається “Siamese network”. Також дану нейронну мережу можна навчати, використовуючи алгоритм бінарної класифікації (англ. *Binary classification*).

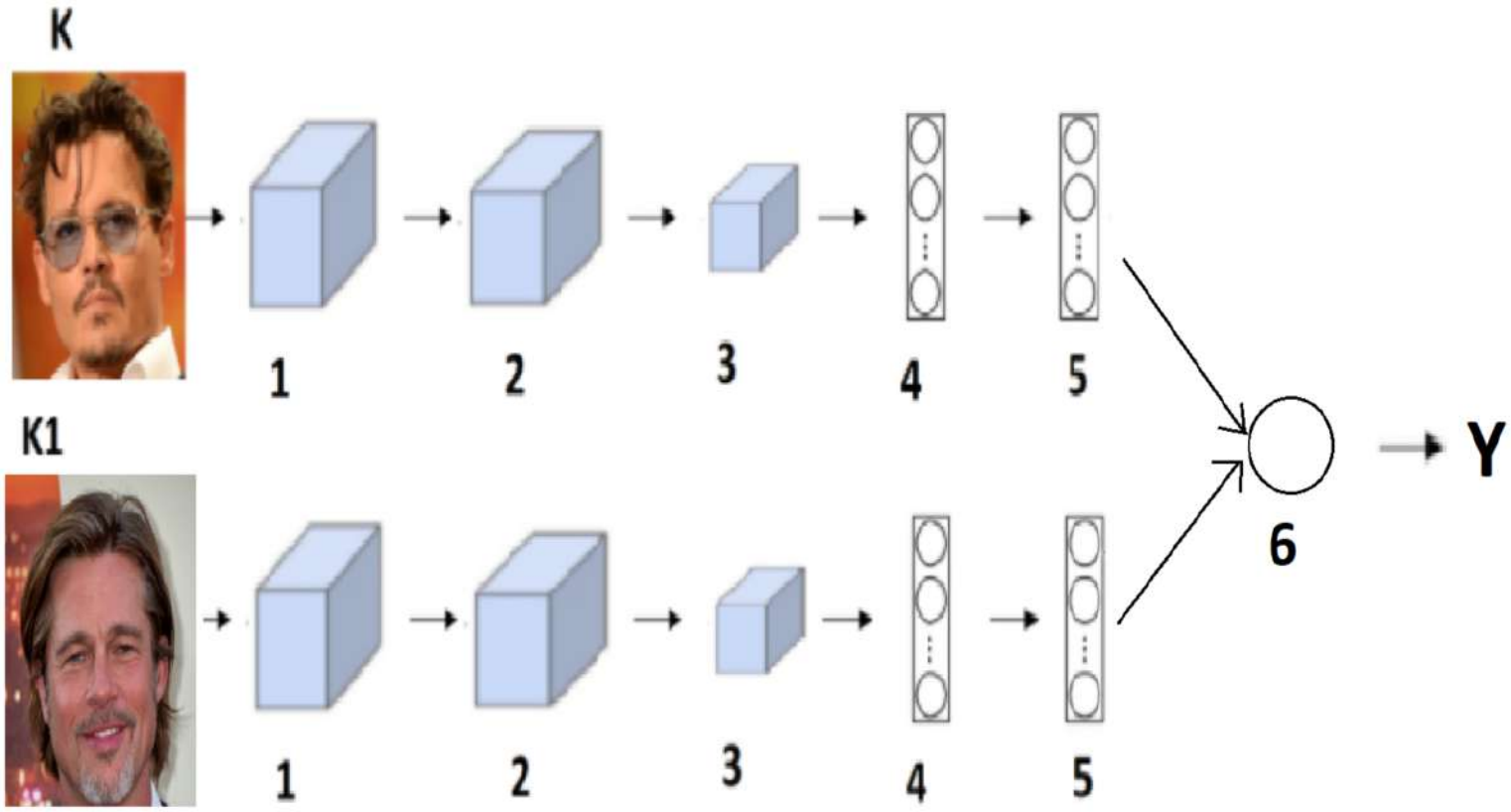

Рис 2.7

Розглянемо Рис 2.7. Як і у попередньому алгоритмі, у нас є нейронна мережа, якій ми передаємо на вхід два зображення - K та K1. Нейронна мережа так само обраховує “кодування” зображення на кроці 5 для K та K1, а потім передає ці кодування як вхідні дані для логістичної регресії (англ. *Logistic regression*).

Логістична регресія робить передбачення і повертає Y. Y дорівнює один, якщо на зображеннях одна і та ж людина, Y дорівнює 0, якщо на зображеннях дві різних людини.

У такий спосіб проблема розпізнавання обличчя може бути розглянута як проблема із розряду бінарної класифікації, це є альтернативним варіантом для Triplet loss функції.

Розглянемо, що саме робить логістична регресія (крок 6 на Рис 2.6). Результат Y (значення 0 чи 1) - це може бути значення sigmoid функції. На вхід sigmoid функція буде приймати різницю між кодуванням K та K1 із кроку 5. Припустимо що довжина закодованого зображення - 128, тоді ми маємо формулу:

$$Y = sigmoid(\sum_{n=1}^{128} | f(K)_n - f(K1)_n |), \text{ де}$$

sigmoid - Сигмоід-функція, f(K) - кодування зображення K, f(K1) - кодування зображення K1. Тобто ми поелементно беремо абсолютну різницю векторів f(K) та f(K1), сумуємо, і потім використовуємо sigmoid-функцію.

Можна думати про елементи векторів f(K) та f(K1) як про “фічі”, які є у вхідних зображеннях. Також до описаної вище функції sigmoid можна додати масив вагів розмірністю 128 та число bias (b), як це відбувається у класичній логістичній регресії:

$$Y = sigmoid(\sum_{n=1}^{128} w_k | f(K)_n - f(K1)_n | + b)$$

Також у дослідженнях був приклад використання іншої функції, яка також застосовувалася в логістичній регресії:

$$Y = sigmoid(\sum_{n=1}^{128} | \frac{(f(K)_n - f(K1)_n)^2}{f(K)_n + f(K1)_n} |)$$

Параметри у верхній та нижній нейронних мережах на Рис 2.7 можуть бути або однаковими, або дуже сильно пов’язаними. Підхід з бінарною класифікацією на практиці також показує хороші результати.

Уявимо, що ми розробляємо систему для доступу співробітників у офіс по скануванню обличчя, використовуючи нашу нейронну мережу. Одним із “прийомів” для оптимізації є заздалегідь обраховані кодування для всіх співробітників із бази даних.

Таким чином, щоб порівняти зображення людини, яка пробує зайти в офіс, не треба буде для кожного співробітника з бази завжди обраховувати кодування зображення, що допоможе сильно зменшити витрати на обчислювальні потужності. Можна навіть не зберігати зображення співробітників у базі даних, можна зберігати тільки їхні “кодування”.

Таку оптимізацію можна використати і для підходу з бінарною класифікацією, і для підходу з Triplet loss функцією.

Приклад тестових даних для навчання нейронної мережі, використовуючи бінарну класифікацію, показано на Рис 2.8. Як ми бачимо, на відміну від Triplet loss функції, у випадку з бінарною класифікацією ми передаємо на три, а два зображення.

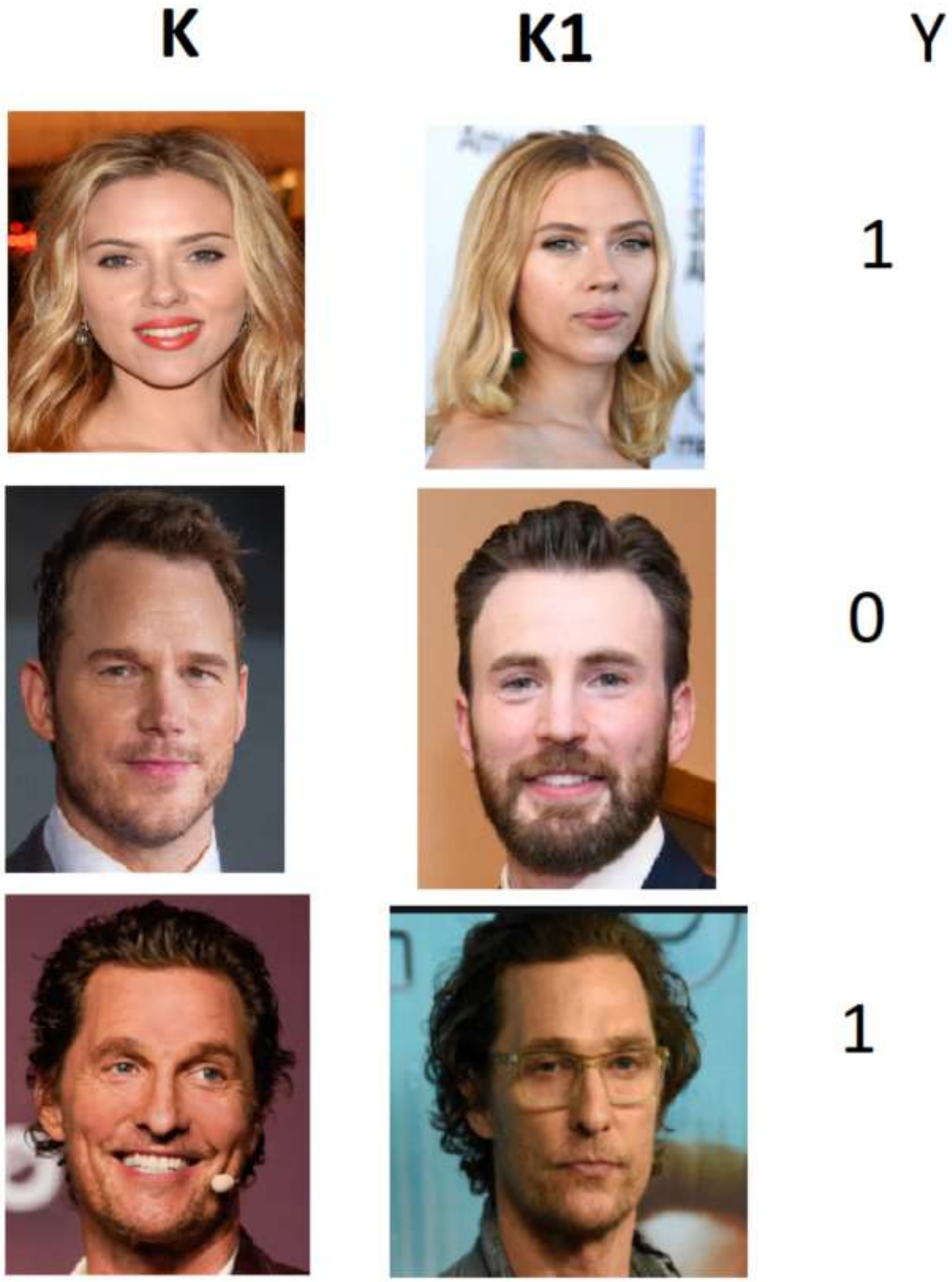

Рис 2.7

Розділ 3: Програмна імплементація алгоритму

Перед тим як імплементувати алгоритм, треба вибрати правильні інструменти. Розглянемо деякі існуючі рішення.

3.1 TensorFlow

Tensorflow - бібліотека для розробки нейронних мереж з відкритим (англ. *Open source*) кодом. На даний момент одне з найпопулярніших рішень. Використовується такими гігантами як CocaCola, AirBnb, Google, Intel, Twitter, Linkedin, Nvidia, Snapchat та ще дуже великою кількістю успішних IT компаній.[15]

Рис 3.1

(https://static.packt-cdn.com/products/9781788397872/graphics/1ebc2a0a-2123-4351-b7e1-eb57f098bafa.png)

API для TensorFlow є для більшості популярних мов програмування: Python, JavaScript, C++, Java, C#, Haskell, R та інші. Багато з них доступні завдяки активному ком'юніті, яке зібралося навколо TensorFlow.

3.2 Keras

Оскільки API для TensorFlow є відносно низькорівневим, для зручності часто використовують **Keras**. Keras - це також рішення з відкритим кодом, яке є більш високорівневою та user-friendly прослойкою між розробником та TensorFlow.

На офіційному сайті сказано, що Keras - створений для людей, а не машин. Має просте і консистентне API та дозволяє мінімізувати кількість дій від користувача.[16]

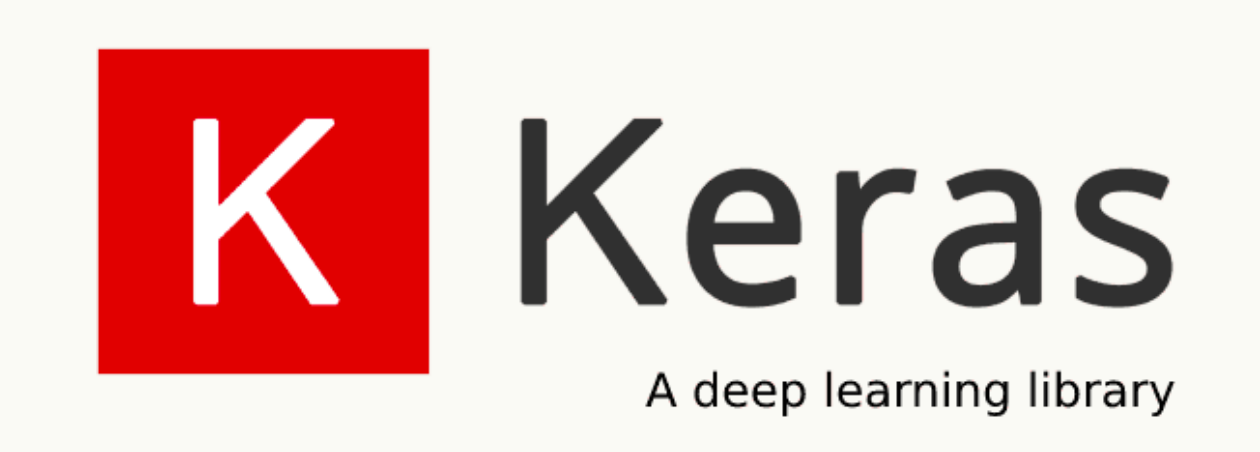

Рис 3.2

(https://miro.medium.com/max/652/1*qllYdaKrZLMhK-nK1zUVEQ.png)

3.3 PyTorch

Ще одним популярним API для роботи з нейронними мережами та машинним навчанням є бібліотека PyTorch. Була розроблена компанією Facebook. PyTorch також є низькорівневим, часто використовується для різних академічних досліджень або для додатків, де треба сильна кастомізація.

Рис 3.3

(https://miro.medium.com/max/691/0*xXUYOs5MWWenxoNz)

3.4 Імплементація

Ми хочемо імплементувати Siamese Network, яка була описана в розділі два. Використовувати будемо Keras, оскільки він має найпростіший API, велике ком’юніті та багато прикладів.

Для тренування Siamese Network можна використати Triplet loss функцію або бінарну класифікацію. Ми будемо використовувати Triplet loss.

Для тренування використаємо Totally Looks Like датасет. Нас цікавить два датасети - left і right. left - містить reference зображення, right - позитивні приклади.

Напишемо функції, які роблять препроцесинг зображень

```
def preprocess_img(filename):
    image_string = tf.io.read_file(filename)
    image = tf.image.decode_jpeg(image_string, channels=3)
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize(image, target_shape)
    return image


def preprocess_triplets(anchor, positive, negative):
    return (
        preprocess_img(anchor),
        preprocess_img(positive),
        preprocess_img(negative),
    )
```

Рис 3.4

Ініціалізуємо refecrence, позитивний та негативний датасети. Також ініціалізуємо тренувальний і валідаційний датасети, як написано на Рис 3.5

```
anchor_images = sorted(
    [str(anchor_images_path / f) for f in os.listdir(anchor_images_path)]
)

positive_images = sorted(
    [str(positive_images_path / f) for f in os.listdir(positive_images_path)]
)

image_count = len(anchor_images)

anchor_dataset = tf.data.Dataset.from_tensor_slices(anchor_images)
positive_dataset = tf.data.Dataset.from_tensor_slices(positive_images)

# To generate the list of negative images, let's randomize the list of
# available images and concatenate them together.
rng = np.random.RandomState(seed=42)
rng.shuffle(anchor_images)
rng.shuffle(positive_images)

negative_images = anchor_images + positive_images
np.random.RandomState(seed=32).shuffle(negative_images)

negative_dataset = tf.data.Dataset.from_tensor_slices(negative_images)
negative_dataset = negative_dataset.shuffle(buffer_size=4096)

dataset = tf.data.Dataset.zip((anchor_dataset, positive_dataset, negative_dataset))
dataset = dataset.shuffle(buffer_size=1024)
dataset = dataset.map(preprocess_triplets)

# Let's now split our dataset in train and validation.
train_dataset = dataset.take(round(image_count * 0.8))
val_dataset = dataset.skip(round(image_count * 0.8))

train_dataset = train_dataset.batch(32, drop_remainder=False)
train_dataset = train_dataset.prefetch(tf.data.AUTOTUNE)

val_dataset = val_dataset.batch(32, drop_remainder=False)
val_dataset = val_dataset.prefetch(tf.data.AUTOTUNE)
```

Рис 3.5

Переконаємося що дані згенерувалися коректно, виведемо на екран кілька триплетів R, P, N - тобто референс зображення, позитивного та негативного, як зображено на Рис 3.6

Рис 3.6

Нашій нейронні мережі треба генерувати “кодування” для кожного зображення із триплета. Для цього ми використаємо вже натреновану мережу ResNet50, яка натренована на ImageNet. Ми приєднаємо до неї декілька Dense слоїв, які ми будемо тренувати.

У ResNet50 ми не хочемо змінювати значення параметрів у натренованих слоях, крім слоя “conv5_block1_out”, як показано на Рис 3.7

```
base_network = resnet.ResNet50(weights="imagenet", input_shape=target_shape + (3,), include_top=False)

flatten = layers.Flatten()(base_network.output)
dense1 = layers.Dense(512, activation="relu")(flatten)
dense1 = layers.BatchNormalization()(dense1)

dense2 = layers.Dense(256, activation="relu")(dense1)
dense2 = layers.BatchNormalization()(dense2)
output = layers.Dense(256)(dense2)

embedding = Model(base_network.input, output, name="Embedding")

trainable = False
for layer in base_network.layers:
    if layer.name == "conv5_block1_out":
        trainable = True
    layer.trainable = trainable
```

Рис 3.7

Додамо функцію, яка рахує дистанцію між парами (R,P) та (R,N).

```
class DistanceLayer(layers.Layer):
    """
    Compute distance between the reference(anchor)
    encoding and the positive encoding as well as distance between anchor and negative encodings
    """

    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def call(self, anchor, positive, negative):
        ap_distance = tf.reduce_sum(tf.square(anchor - positive), -1)
        an_distance = tf.reduce_sum(tf.square(anchor - negative), -1)
        return (ap_distance, an_distance)

anchor_input = layers.Input(name="anchor", shape=target_shape + (3,))
positive_input = layers.Input(name="positive", shape=target_shape + (3,))
negative_input = layers.Input(name="negative", shape=target_shape + (3,))

distances = DistanceLayer()(
    embedding(resnet.preprocess_input(anchor_input)),
    embedding(resnet.preprocess_input(positive_input)),
    embedding(resnet.preprocess_input(negative_input)),
)

siamese_network = Model(
    inputs=[anchor_input, positive_input, negative_input], outputs=distances
)
```

Рис 3.8

Запустивши код, переконуємося що Loss-функція зменшується, відповідно нейронна мережа тренується коректно.

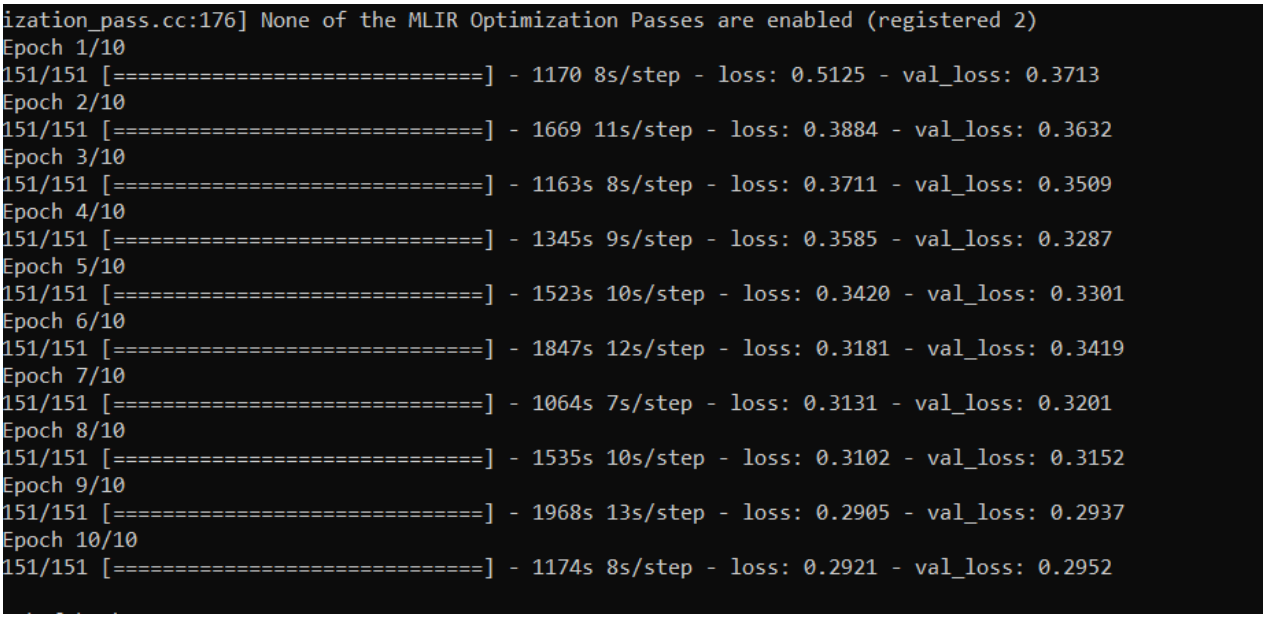
ization_pass.cc:176] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/10
151/151 [==============================] - 1170 8s/step - loss: 0.5125 - val_loss: 0.3713
Epoch 2/10
151/151 [==============================] - 1669 11s/step - loss: 0.3884 - val_loss: 0.3632
Epoch 3/10
151/151 [==============================] - 1163s 8s/step - loss: 0.3711 - val_loss: 0.3509
Epoch 4/10
151/151 [==============================] - 1345s 9s/step - loss: 0.3585 - val_loss: 0.3287
Epoch 5/10
151/151 [==============================] - 1523s 10s/step - loss: 0.3420 - val_loss: 0.3301
Epoch 6/10
151/151 [==============================] - 1847s 12s/step - loss: 0.3181 - val_loss: 0.3419
Epoch 7/10
151/151 [==============================] - 1064s 7s/step - loss: 0.3131 - val_loss: 0.3201
Epoch 8/10
151/151 [==============================] - 1535s 10s/step - loss: 0.3102 - val_loss: 0.3152
Epoch 9/10
151/151 [==============================] - 1968s 13s/step - loss: 0.2905 - val_loss: 0.2937
Epoch 10/10
151/151 [==============================] - 1174s 8s/step - loss: 0.2921 - val_loss: 0.2952

Рис 3.9

Висновки

Нейронні мережі - це технологія, яка на даний момент дуже активно розвивається та рухається вперед.

Було детально розглянуто існуючі типи нейронних мереж, як вони з’явилися та чим відрізняються. Розглянуто типи задач, які нейронні мережі можуть вирішити.

Потім було детально розглянуто алгоритм розпізнавання обличчя та різні варіанти його імплементації.

У кінці був невеликий огляд існуючих бібліотек для написання нейронних мереж та імплементація нейронної мережі, яка натренована для розпізнавання облич.

Список використаних джерел

1. Що таке нейрон - https://dovidka.biz.ua/shho-take-neyron/
2. How many neurons in the human brain - https://www.verywellmind.com/how-many-neurons-are-in-the-brain-2794889
3. Artificial Neural Network - https://www.investopedia.com/terms/a/artificial-neural-networks-ann.asp
4. Neural networks history - https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html
5. History of neural networks - https://medium.com/analytics-vidhya/brief-history-of-neural-networks-44c2bf72eec
6. What is an artificial neuron- http://neuralnetworksanddeeplearning.com/chap3.html#regularization
7. Why do Neural Networks Need an Activation Function? - https://towardsdatascience.com/why-do-neural-networks-need-an-activation-function-3a5f6a5f00a
8. A Gentle Introduction to the Rectified Linear Unit (ReLU) - https://towardsdatascience.com/why-do-neural-networks-need-an-activation-function-3a5f6a5f00a
9. What is an artificial neuron and why does it need an activation function?- https://towardsdatascience.com/what-is-an-artificial-neuron-and-why-does-it-need-an-activation-function-5b4c1e971d80

10. Supervised vs Unsupervised Learning - https://www.guru99.com/supervised-vs-unsupervised-learning.htm
11. Feedforward Neural Networks - https://brilliant.org/wiki/feedforward-neural-networks/
12. Fully Connected vs Convolutional Neural Networks - https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5
13. Understanding recurrent neural networks - https://builtin.com/data-science/recurrent-neural-networks-and-lstm
14. Convolutional Neural Networks - https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939
15. TensorFlow - https://www.tensorflow.org/
16. Keras - https://keras.io/