Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

Розробка нейронної мережі по розпізнаванню жестової мови

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи

к.т.н., доц. Жежерун О.П.

_____

(підпис)

“____” _____ 2021 р.

Виконала студентка БП ІПЗ-4

Гальченко М.О.

“____” _____ 2021 р.

Київ 2021

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики
ЗАТВЕРДЖУЮ
Зав. кафедри мультимедійних систем,
доцент, к.т.н _____ Жежерун О.П.
(підпис)
«____» _____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студентці Гальченко Марії Одіссеївні
Факультету інформатики 4 р.н. бакалаврської програми
ТЕМА: Розробка нейронної мережі по розпізнаванню жестової мови

Зміст текстової частини до курсової роботи:
- Індивідуальне завдання
- Вступ
- Огляд теоретичного матеріалу і здійснення дослідження
- Опис реалізації програмного продукту
- Висновки

Список літератури
Додатки (за необхідністю)
Список посилань

Дата видачі «___» _____ 2021 р. Керівник_____

Завдання отримав _____

Тема: Розробка нейронної мережі по розпізнаванню жестової мови

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	10.11.2020	
2.	Огляд літератури за темою роботи	17.12.2020	
3.	Проведення досліджень	27.01.2021	
4.	Опис результатів досліджень	23.02.2021	
5.	Аналіз отриманих результатів з керівником	05.03.2021	
6.	Коригування роботи	25.03.2021	
7.	Створення презентації та написання доповіді	11.04.2021	
9.	Захист курсової роботи	19.04.2021	

Студентка Гальченко М.О.

Керівник Жежерун О.П.

“_____” _____

Зміст

Анотація

У даній роботі було проведено аналіз систем розпізнавання жестової мови та алгоритмів обробки даних. Вивчені найпопулярніші бібліотеки для створення нейронних мереж. На основі дослідженної інформації спроектовано та створено нейронну мережу, призначену для розпізнавання жестової мови на зображеннях і відео. По результатам проведених досліджень було прийнято рішення використовувати бібліотеку “комп’ютерного зору” OpenCV, PyTorch для створення нейронної мережі та Onnx для експорта готової нейронної мережі. Створена нейронна мережа здатна розпізнавати жестову мову та перекладати її в український алфавіт. Реалізація програмного забезпечення розроблено на мові Python з використанням бібліотек.

Вступ

У наш час комунікація людей з обмеженими можливостями досі є актуальною проблемою. По даним Всесвітньої організації охорони здоров'я близько 460 мільйонів людей страждають від втрати слуху або його порушення. Згідно до статистики Українського товариства глухих за 2017 рік [1], в Україні більше 50 000 громадян мають різноманітні вади слуху та мовлення. Проте, дане число є набагато більшим, адже це лише ті люди, що зареєстровані в базі. По неофіційним даним - більше 3 мільйонів. По даним Всесвітньої організації глухих на 2021 рік, число носіїв Української жестової мови складає близько 200 000 тисяч чоловік. [2]

Згідно з дослідженою інформацією, для вивчення та ознайомлення з українською жестовою мовою, існує лише один сайт: spreadthesign. Проте, навіть він не є повним і лінгвістично достовірним. Саме тому, у людей, що використовують жестову мову в Україні, даний процес може викликати багато труднощів.

Метою даного проекту є розробка програмного забезпечення, здатного розпізнавати українську жестову мову. Дана робота реалізована у середовищі розробки Jupyter Notebook. Для написання додатку та створення зручного користувацького інтерфейсу використовується мова Python. Для роботи з алгоритмами комп'ютерного зору, обробкою зображень та експортом моделей використовується бібліотека OpenCV та ONNX. Нейронна мережа розроблена за допомогою фреймворку для машинного навчання PyTorch.

Робота була поділена на наступні розділи:

У першому розділі було проведено аналіз предметної області та описано структуру жестів.

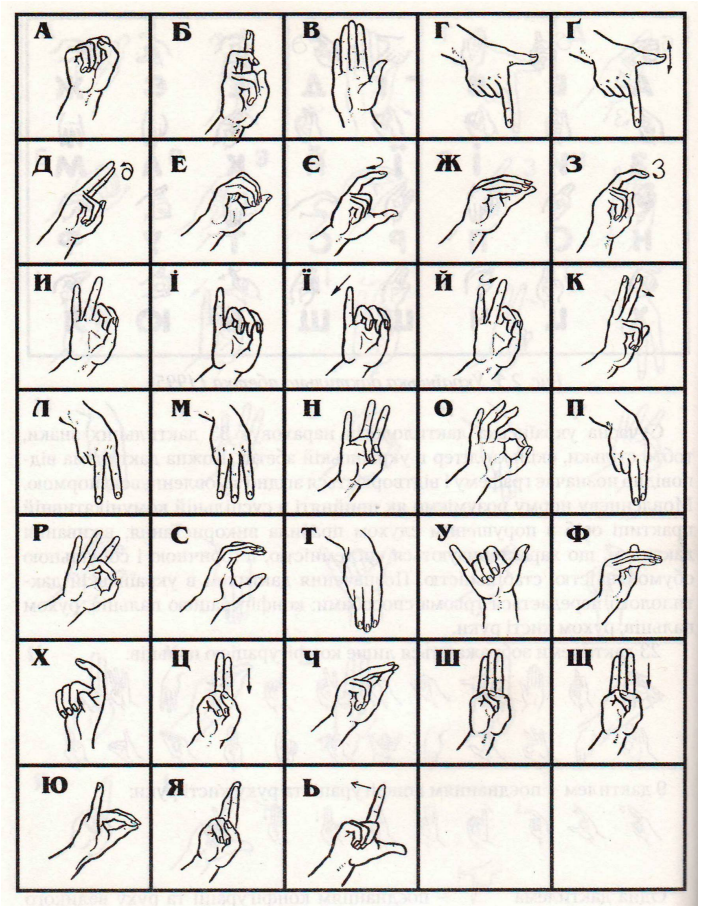
У другому розділі проведено огляд основних видів, задач, алгоритмів та підходів до машинного навчання. Наведені приклади їх використання у реальному житті. Крім того, другий розділ містить інформацію про обраний алгоритм та обґрунтування його вибору.

У третьому розділі детально описано реалізацію створення нейронної мережі та програмного застосунку. Також додані приклади роботи програми.

1. Аналіз предметної області. Постановка завдання

1.1. Загальна інформація про жестову мову

У наш час у всьому світі нараховується близько 150 жестових мов.[3] Відповіді на те, як з’явилася така кількість жестових мов немає й досі. Найкращим прикладом можна навести Нікарагуанську мову жестів. Вона з’явилася у 80-ті роки XX сторіччя, коли відкрили школу для глухих у Нікарагуа. Діти для того, щоб мати змогу спілкуватись на перервах, розробили власну мову жестів та правила. Вчені протягом двох-трьох сторіч вивчали, як розвивалась граматика та лексика та вияснили, що схожий шлях створення мови, пройшло більшість жестових мов. [4] На даному малюнку представлена українська дактильна абетка.



У сучасній Україні жестова мова з’явилась у часи заснування перших громад та спеціальних шкіл для глухих та слабочуючих. 1830 року було відкрито Львівську школу для глухих дітей, а 1843-го – Одеську. [5]

1.2. Структура жестової мови

Умовно будь-який жест можна розділити на 4 складові:

- рух руки у момент виконання жесту;
- конфігурація - форма руки, що виконує жест;
- орієнтація - рух руки по відношенню до корпусу тіла;
- локалізація - розташування жесту.

Крім того, не менш важливими елементами є міміка обличчя та артикуляція.

Поширеною ситуацією є випадок, коли поняття вже є, проте жест на нього ще не вигадали. У таких випадках використовується дактильна абетка, яку можна назвати “буквами” жестової мови.

1.3. Постановка задачі

- Проаналізувати предметну область та можливі аналоги розробки.
- Проаналізувати існуючі методи захоплення, відстеження та розпізнавання жестів.
- Проаналізувати алгоритми обробки даних.
- Розробити застосунок для розпізнавання жестової мови дактильної абетки української мови.
- Протестувати додаток і провести аналіз отриманих результатів.

2. Теоретичні відомості. Машинне навчання

2.1. Основні види та задачі машинного навчання

Кожна людина має індивідуальні особливості власного тіла. У випадку написання нейронної мережі по розпізнаванню жестової мови, нас цікавлять руки людей, які мають безліч характеристик: розмір руки, довжина пальців, їх товщина, гнучкість і так далі. Саме морфологічні особливості можуть повпливати на успішність розпізнавання жестів. Враховуючи це, необхідно вміти максимально точно визначати жест людини.

Більшість задач, що вирішуються за допомогою машинного навчання, відносяться до одного з видів: supervised learning [6] (навчання з учителем) або unsupervised learning [7] (без учителя). “Учитель” у розумінні машинного навчання - це задіяність безпосередньо людини у процес обробки інформації. В обох випадках навчання машини є вихідні дані, які вона має проаналізувати та знайти по ним певні закономірності. Відмінність двох видів лише у тому, що при навчанні з учителем є ряд гіпотез, які необхідно або спростувати, або підтвердити.

Машинне навчання з учителем - напрямок, що об’єднує методи та алгоритми побудови моделей на основі великої кількості прикладів, що включають в себе пари по типу “об’єкт - відповідь”. У такому випадку, необхідно знайти функціональну залежність відповідей від параметрів об’єктів і побудувати такий алгоритм, що зможе по вхідному об’єкту, на виході видавати коректну відповідь. Наприклад, ми маємо дані про тисячі квартир в одному з Київських районів (Осокорки): розміри, кількість кімнат, час побудови, поверх, інфраструктура, транспортна розв’язка, ціна і так далі. Наша задача - створити модель, що по заданим критеріям розраховує ринкову вартість. Це ідеальний приклад використання машинного навчання з учителем, коли ми маємо відомі вихідні дані (кількість квартир та їх параметри) та сформовану відповідь по кожній з квартир (їх вартість). Ще

одним яскравим прикладом даного виду машинного навчання є задача, коли необхідно визначити чи є лист спамом, проаналізувавши текст.

Основні задачі машинного навчання з учителем:

- Задача класифікації характерна тим, що має скінченну кількість відповідей (в більшості випадків у форматі “Так” або “Ні”). Наприклад, чи є на фотографії квіти, чи є зображення рукою людини, чи хвора людина на туберкульоз.
- Задача регресії характерна тим, що допустимою відповіддю можуть бути числовий вектор або дійсні числа. Наприклад, ціна торта, квартири, прибуток кіоска за наступний місяць.

У випадку машинного навчання без учителя готових відповідей зазвичай немає. Наприклад, ми маємо інформацію про зріст, об’єм грудної клітки та ваги тисячі людей. Ці дані ми маємо розподілити по 3-5 групах та відшити футболки правильного розміру на кожну групу. Таким чином, усі дані необхідно розділити на 3-5 кластерів. Ще одним прикладом є ситуація, коли об’єкти вибірки мають тисячі різноманітних параметрів. Тоді, виникають складнощі з графічним відображенням даної вибірки. У такій ситуації кількість параметрів зменшують до двох, трьох і з’являється можливість візуалізації об’єктів на площині або у 3д.

Основні задачі машинного навчання без учителя:

- Задача кластеризації розділяє дані на групи(кластери). Наприклад, віднести комп’ютерну гру до якоїсь категорії(шутер, стратегія, аркади, симулятори, головоломки і так далі).
- Задача зменшення розмірності характерна тим, що має на меті зведення великої кількості параметрів до мінімальної кількості для їх подальшої візуалізації (data compression).
- Задача виявлення аномалій подібна до задачі класифікації. Аномалії - це доволі рідкісне явище, тому приклади, на основі яких можна навчити модель на вияв таких об’єктів дуже мало або їх може навіть не існувати. Саме тому задача класифікації у такій ситуації не спрацює. У

якості прикладу, можна навести вияв шахрайських дій із банківськими картками.

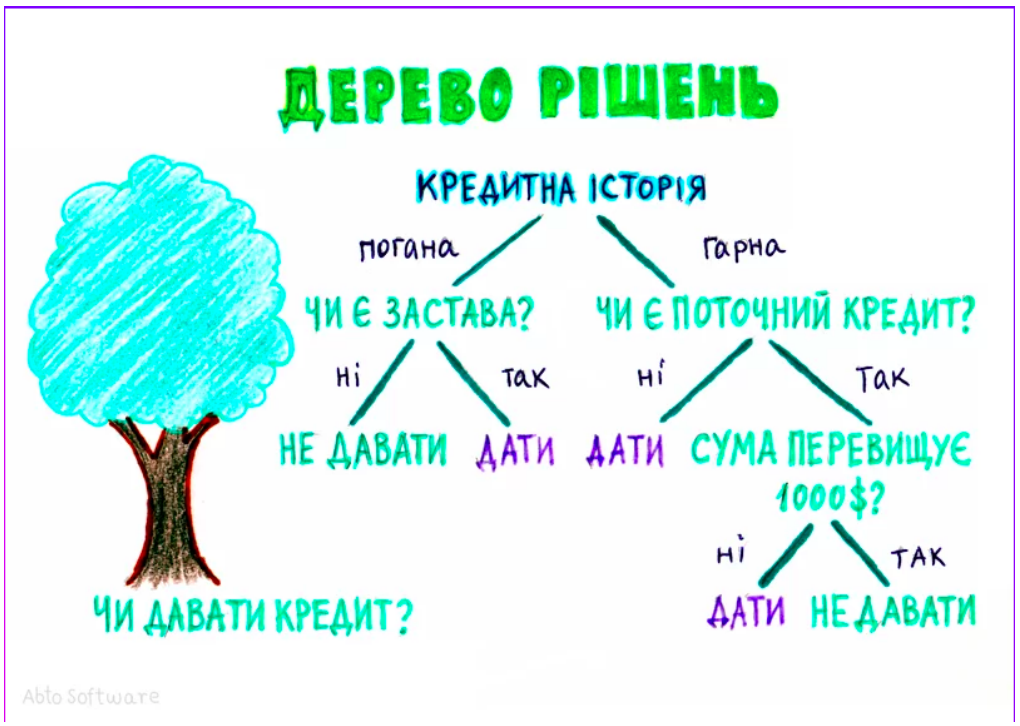
2.2. Алгоритми та підходи до машинного навчання

Для того щоб запобігти проблем з розпізнаванням жестів через недостатню точність, виникнення конфліктів або неоптимальну кількість розрахункових ресурсів, було вирішено провести аналіз алгоритмів машинного навчання та обрати найкращий та найбільш корисний підхід.

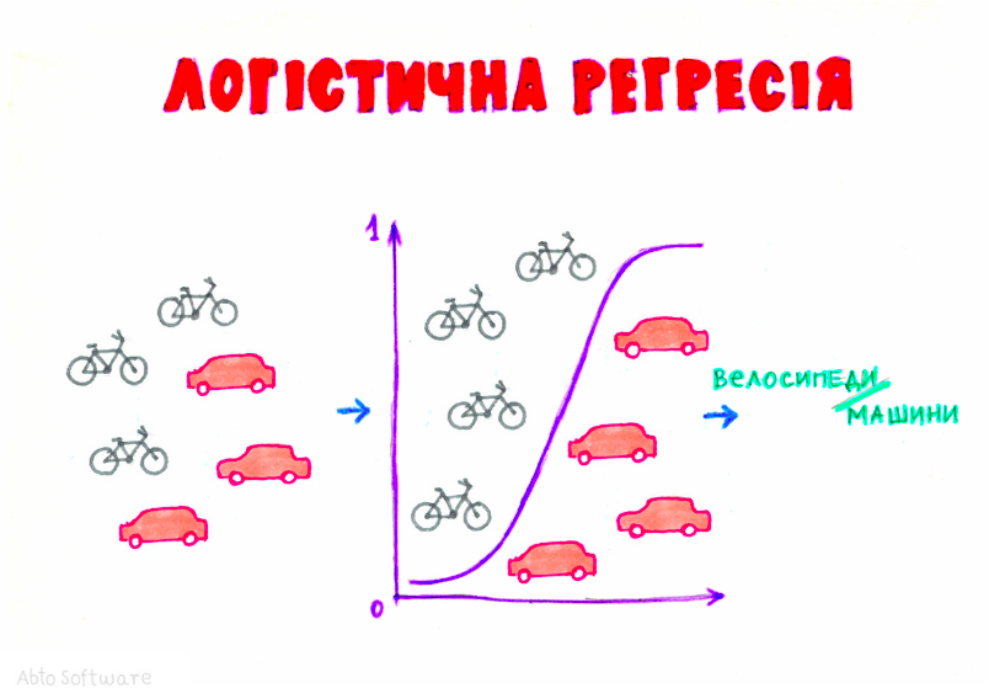
Підходи до задач машинного навчання - певна концепція або точка зору на сам процес навчання, що призводить до створення набору гіпотез та припущень, на основі яких потім будується модель. Розділяти алгоритми на підходи досить умовна річ. За допомогою різних підходів можна прийти до однакової остаточної моделі, проте методи її навчання можуть сильно відрізнятися.

- Наївний байєсів класифікатор походять від теореми Байєса, у якій функції розглядаються як незалежні (суворе припущення). Практично застосовуються для: виявлення спаму електронної пошти, розпізнавання обличчя або інших частин на зображенні, визначення емоційного забарвлення текстів, автоматичне визначення тематичних рубрик новин. Наприклад, раніше за допомогою наївного байєса були розроблені спеціальні фільтри від спаму. Машина підраховувала кількість входжень слова “приз” у звичайному листі та у листі зі спамом. Після чого усі ймовірності перемножувалися за формулою Байєса, результати підсумовувалися і програма могла визначати листи зі спамом.

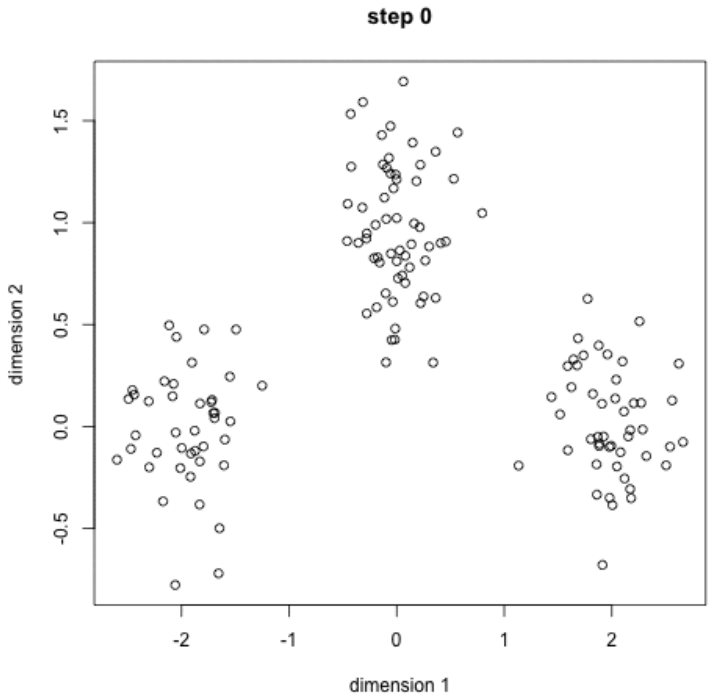
- Дерево рішень - метод прийняття рішень за допомогою використання деревовидного графа. Дерево будується на основі мінімальної кількості питань, що повинні мати однозначну відповідь (“Так ” або “Ні”). Даний алгоритм систематизує та структурує проблему. Відповівши почергово на всі запитання, ми можемо прийти до логічного та правильного висновку.

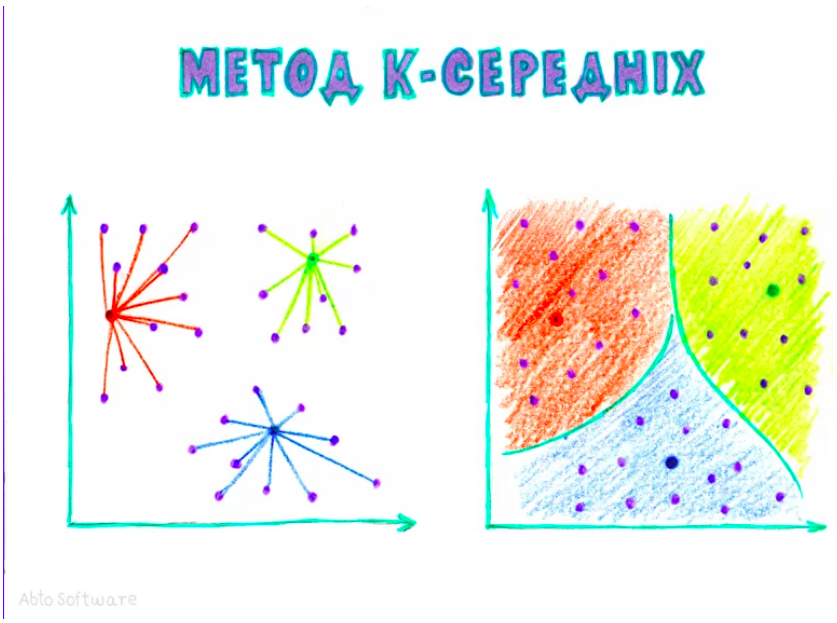


- Логістична регресія - метод визначення залежностей між змінними, коли одна з них категорійно залежна, а інші незалежні. На практиці такий алгоритм використовують у випадках, коли за допомогою статичного методу з однією або декількома незалежними змінними, ми можемо передбачити певні події. Наприклад, передбачення успіху проведення певної рекламної кампанії, прогнозування прибутку від конкретного товару, прогноз шансу землетрусу в конкретну дату і так далі. Для задач бінарної класифікації, наприклад, якщо ми хочемо визначити до якого з двох класів належить об’єкт, логістична регресія є ідеальним варіантом вирішення. Функція логістичної регресії перетворює значення у число в межах від нуля до одиниці, передбачаючи належність до одного чи другого класу.



- Метод k-середніх - є одним із найбільш популярних алгоритмів кластеризації. Він заключається в тому, що точки набору даних ділять на k кластерів ґрунтуючись на найближчих середніх значеннях. У кожному кластері відстань між точками має бути обов’язково мінімальною.





Методи, що були перераховані та описані вище - вважаються класичними у машинному навчанні. Проте, найбільш популярними та сучасними на сьогоднішній день є саме нейронні мережі, що можуть виконувати будь-які задачі, включаючи задачі перерахованих вище алгоритмів.

Будь-яка нейронна мережа - це набір нейронів та зв’язків між ними. Нейрони можна охарактеризувати, як функцію з багатьма входами і одним виходом. Задача нейрона - зібрати числа з усіх входів, застосувати над ними функцію і відправити обрахований результат на вихід. Зв’язки - канали, по яким нейрони відправляють один одному числа. Кожному зв’язку присвоюють певну вагу - параметр, що можна описати, як міцність зв’язку, тобто простішими словами вага або збільшує або зменшує силу сигналу на з'єднанні. Наприклад, коли через зв’язок з вагою 0,5 проходить число 20, воно перетворюється у 10. Нейрони самі по собі не розуміють, що саме до нього прийшло і підсумовує усі числа. Отже, для того, щоб керувати на які входи нейрону реагувати, а на які ні, і потрібна вага. Для того, щоб безліч нейронів не знаходились у стані хаоса, нейрони вирішили пов’язувати по шарах. У кожному шарі нейрони ніяк не пов’язані між собою, проте вони пов’язані з

нейронами попереднього та наступного шара. Таким чином, дані мають один конкретний напрямок від входів першого шару до виходів останнього.

2.3. Приклади використання у реальному житті

На сьогоднішній день, за рахунок того, що з’являться все більше і більше технологій, з такою ж швидкістю люди стають усе більш лінивими. Саме тому машинне навчання та штучний інтелект із шаленими обертами стає все більш популярним. Гарні спеціалісти по машинному навчанню мають великий попит на сучасному ринку праці. У нашому світі, є безліч задач, які може вирішити машинне навчання. Найвідоміші приклади його використання:

1. Діагностика захворювань:

Об’єктами в такому випадку є пацієнти, а властивості - це всі симптоми, результати аналізів, історія хвороб, рівень речовин у крові, вік, вага, стать. Маючи повну інформацію про свого пацієнта, її можна завантажити у програму з машинним навчанням та вирішити велику кількість задач, наприклад: визначити вид захворювання, обрати найкраще лікування, поставити прогноз розвитку хвороби та можливих ускладнень, виявити синдроми. Швидкості роботи такої програми може лише позаздрити будь-який лікар. У якості приклада, можна навести діагностику рака шкіри або діабетичну ретинопатію. У цьому допомагає машинне навчання - комп’ютер досліджує більше 125 тисяч знімків судинної оболонки та може виявити хворобу на ранній стадії.

2. Кредитний скоринг:

Коли банк приймає рішення чи варто видавати кредит клієнту, враховується рейтинг його платоспроможності. На основі багатьох параметрів (вік, зарплатня, кредитна історія і так далі) штучний інтелект після їх обробки видає позитивне або негативне рішення.

3. Розпізнавання картинок або облич:

Останнім часом дані системи є дуже популярними, адже їх активно використовують у більшості соціальних мереж (Інстаграм, ВК, Фейсбук, Тік-Ток). Для того, щоб накладувати фільтри, робити відмітки друзів на фото,

шукати схожих людей і так далі. Камери, що можуть розпізнавати обличчя у місті також є прикладом використання машинного навчання.

4. Розпізнавання мови:

Розпізнавання мови та негайна конвертація її у друкований текст. Голосовий ввід, пошук, перекладачі, Сірі, Аліса - те, чим ми зазвичай користуємося кожного дня є яскравим прикладом використання машинного навчання.

5. Геймдев:

У ході розробки ігор необхідно враховувати велику кількість нюансів. Якщо гра має “відкритий” світ, треба опрацювати багато деталей: прорахувати всі можливі дії гравця, штучного інтелекту та неігрових персонажів, анімація, зміни у навколишньому світі, сеттінг. У такій обширній роботі машинне навчання задіюють по максимуму.

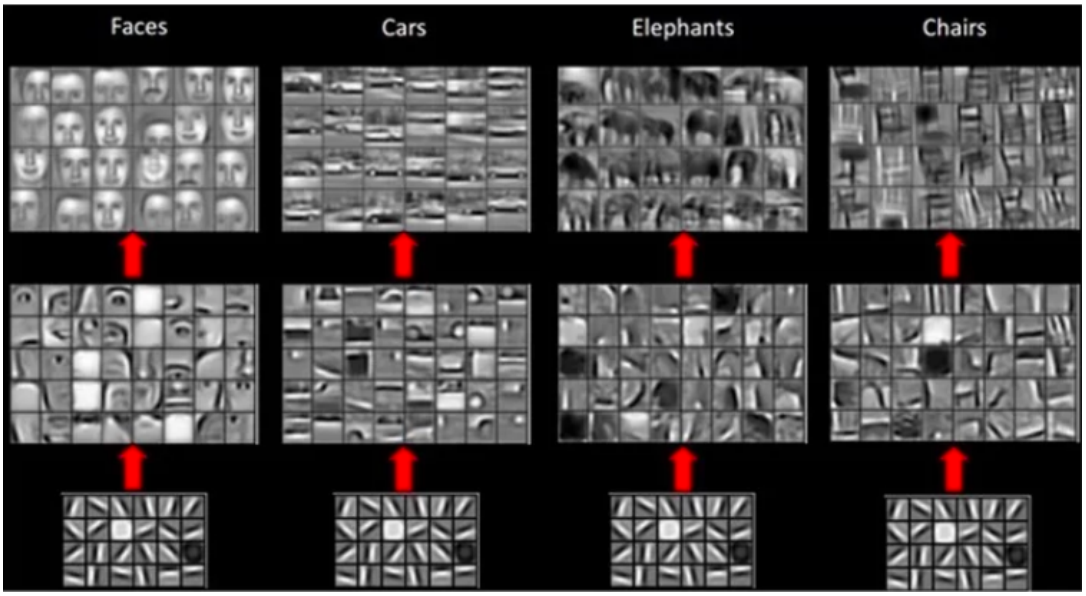
6. Рекомендації:

На основі наших пошукових запитів, відвіданих профілів інших користувачів, вподобанням, прослуханої музики кожного дня формуються рекомендації.

2.4. Обґрунтування вибору

Після детального дослідження більшості існуючих алгоритмів машинного навчання, необхідно обрати найкращий, що буде повністю задовольняти програмний застосунок. Таким чином, було вирішено використовувати згорткову нейронну мережу(CNN) [8]. Сьогодні згорткові нейронні мережі використовуються для багатьох цілей: розпізнавання різних частин, пошук об’єктів на фото чи відео, створення різноманітних ефектів на відео, покращення якості фото і так далі. Можна зробити висновок, що CNN застосовують будь-де, де є фото чи відео.

Основним завданням моєї нейронної мережі має бути її хороша здатність розпізнавати конкретний жест. У випадку використання згорткової нейронної мережі, ми проганяємо нашу нейронну мережу через величезну кількість фото жестів. CNN автоматично проставляє велику вагу тим елементам фото, які вона бачила частіше усього. На виході ми поставимо найпростіший перцептрон, що буде визначати які комбінації взагалі активувались і казати, якому жесту більш за все відповідає положення руки на фото чи відео.

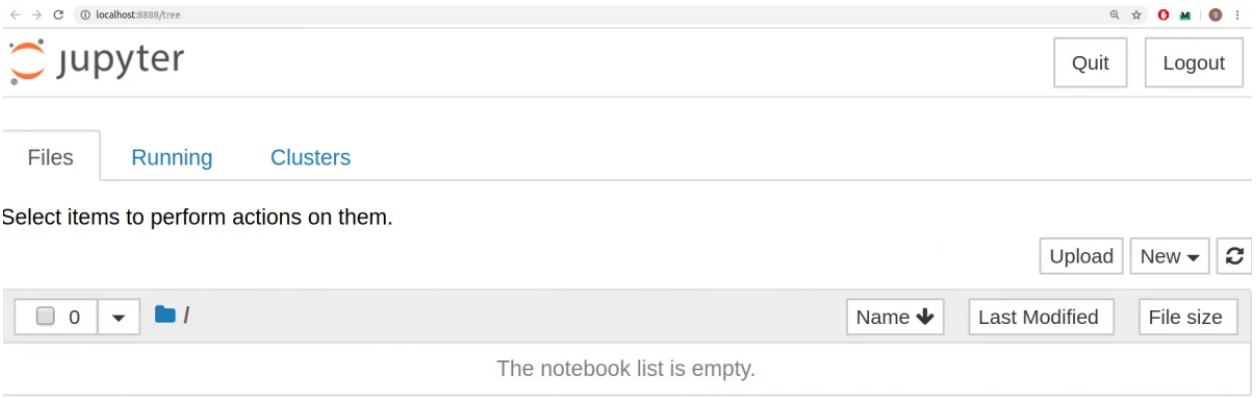


3. Розробка нейронної мережі

3.1. Засоби розробки

При розробці програмного застосунку використовувались наступні технології: Python, OpenCV, PyTorch, ONNX.

Враховуючи те, що для роботи над проектом ми маємо використовувати багато бібліотек та модулів, мною була обрана мова розробки Python. Python - найпопулярніша мова розробки нейронних мереж. Вона неймовірно проста, лаконічна, з найменшими витратами часу та сил ми маємо можливість розробляти складні алгоритми.



Для зручної роботи над проектом було вирішено використовувати Jupyter Notebook - потужний інструмент для розробки і представлення проектів, написаних на мові Python. Підключається дана утиліта через термінал після завантаження бібліотеки Jupyter. Усі дії виконуються віддалено на сервері у браузері.

```
pip3 install jupyter
```

У ньому зручно об’єднувати код і вивід у вигляді одного документу, що включає у себе текст, функції та візуалізацію. Процес розробки є неймовірно зручним, адже ви можете покроково виконувати кожну частинку коду та відразу виводити результат його виконання окремими блоками.

```
In [2]: import torch

In [7]: from torch.utils.data import Dataset
        from torch.autograd import Variable
        import torch.nn as nn
        import numpy as np
        import torch
```

Для створення глибинної нейронної мережі було прийняте рішення використовувати сучасну бібліотеку на основі Python для глибинного навчання - PyTorch. Ідеальна бібліотека глибинного навчання повинна бути простою у використанні та освоєнні, потужною у розрахунках та швидкості обробки даних, плюс до всього, точною у всіх отриманих результатах. Після дослідження аналогів, таких як: TensorFlow та Keras, було вирішено використовувати фреймворк PyTorch.

Найпопулярнішим продуктом комп'ютерного зору є бібліотека з відкритим кодом OpenCV (Open Source Computer Vision Library). Дана бібліотека реалізована на C++, проте може використовуватись у Python, Java, Ruby і так далі. OpenCV має більше 2500 оптимізованих алгоритмів для обробки зображень.

ONNX (Open Neural Network Exchange) - відкрита бібліотека, що використовується для побудови глибинних нейронних мереж. ONNX дозволяє навчати моделі, а потім експортовувати їх в іншу середу розробки.

3.2. Розробка застосунку

3.2.1. Підготовка даних для класифікації жестової мови

На цьому етапі, ми застосовуємо унітарне кодування всіх міток і перетворюємо наші дані в тензори PyTorch. Для того, щоб наша програма розпізнавала “незвичні” вводи, такі як перевернута або викручена рука ми робимо приріст даних. Зрештою, ми отримаємо алгоритмічний метод доступу до зображень і міток в наборі даних.

```
def get_label_mapping():
    mapping = list(range(33))
    mapping.pop(9)
    return mapping
```

Мітки варіюються від 0 до 33. Проте не включаючи букв Ж(9) і Я(33). Отже, є лише 31 значення міток. Метод *get_label_mapping*() трансформує мітки набору даних[0, 33] в індекси букв [0, 33].

Після цього ми додаємо метод добування міток с CSV файлу. При наступному виводі, кожен рядок починається з label і йде значення 784(зображення 28x28) пікселів.

```
def read_label_samples_from_csv(path: str):
    """
    Assumes first column in CSV is the label and subsequent 28^2 values
    are image pixel values 0-255.
    """
    mapping = SignLanguageMNIST.get_label_mapping()
    labels, samples = [], []
    with open(path) as f:
        _ = next(f)  # skip header
        for line in csv.reader(f):
            label = int(line[0])
            labels.append(mapping.index(label))
            samples.append(list(map(int, line[1:])))
    return labels, samples
```

Методи: *init* ініціалізує тримача даних та *len* допомагає визначити, коли варто припинити ітерацію даних.

```
    def __init__(self,
                 path: str="data/sign_mnist_train.csv",
                 mean: List[float]=[0.485],
                 std: List[float]=[0.229]):
        labels, samples = SignLanguageMNIST.read_label_samples_from_csv(path)
        self._samples = np.array(samples, dtype=np.uint8).reshape((-1, 28, 28, 1))
        self._labels = np.array(labels, dtype=np.uint8).reshape((-1, 1))

        self._mean = mean
        self._std = std

    def __len__(self):
        return len(self._labels)
```

Метод *getitem* повертає словник з міткою та зразком.

```
    def __getitem__(self, idx):
        transform = transforms.Compose([
            transforms.ToPILImage(),
            transforms.RandomResizedCrop(28, scale=(0.8, 1.2)),
            transforms.ToTensor(),
            transforms.Normalize(mean=self._mean, std=self._std)])

        return {
            'image': transform(self._samples[idx]).float(),
            'label': torch.from_numpy(self._labels[idx]).float()
        }
```

За допомогою класса SignLanguageMNIST ініціалізуємо набір даних та перетворюємо набір даних в DataLoader.

```
def get_train_test_loaders(batch_size=32):
    trainset = SignLanguageMNIST('data/sign_mnist_train.csv')
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True)

    testset = SignLanguageMNIST('data/sign_mnist_test.csv')
    testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffle=False)
    return trainloader, testloader
```

3.2.2. Створення класифікатора жестової мови

У даному розділі ми створюємо шестирівневу нейронну мережу та оптимізатор. Оцінюємо втрати даних та оптимізуємо функцію втрат.

Спочатку ми визначаємо нейронну мережу, що складається з трьох згорток(згорткових шарів), за якими слідують три повноз’єднані шари.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 6, 3)
        self.conv3 = nn.Conv2d(6, 16, 3)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 48)
        self.fc3 = nn.Linear(48, 24)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Ініціалізуємо нейронну мережу, визначаємо функцію втрат і визначаємо гіперпараметри оптимізації.

```
def main():
    net = Net().float()
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)

    trainloader, _ = get_train_test_loaders()
    for epoch in range(2):  # loop over the dataset multiple times
        train(net, criterion, optimizer, trainloader, epoch)
    torch.save(net.state_dict(), "checkpoint.pth")
```

Ми визначаємо період для ітерації там, де кожен зразок використовувався хоча б один раз. Параметри моделі будуть зберігатись у файлі checkpoint.pth .

Витягаємо image та label і “загортаємо” їх у *Variable* PyTorch. Попередньо пройдемося, виконаємо зворотне поширення по нейронній мережі та втратам.

```
def train(net, criterion, optimizer, trainloader, epoch):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs = Variable(data['image'].float())
        labels = Variable(data['label'].long())
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels[:, 0])
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 100 == 0:
            print('[%d, %5d] loss: %.6f' % (epoch, i, running_loss / (i + 1)))
```

Запускаємо навчання. Для зменшення втрат бажано запускати на кількість періодів більше 10. Тому запустимо на 12 періодів. Вивід виглядає так:

```
[0,     0] loss: 3.234514
[0,   100] loss: 3.213096
[0,   200] loss: 3.190615
[0,   300] loss: 2.961571
[0,   400] loss: 2.658706
[0,   500] loss: 2.376268
[0,   600] loss: 2.120140
[0,   700] loss: 1.918828
[0,   800] loss: 1.751443
[1,     0] loss: 0.687203
[11,   100] loss: 0.008328
[11,   200] loss: 0.008543
[11,   300] loss: 0.009508
[11,   400] loss: 0.008793
[11,   500] loss: 0.009405
[11,   600] loss: 0.008773
[11,   700] loss: 0.008325
[11,   800] loss: 0.008403
```

Остаточний результат втрат - 0.008403, що майже втричі менше початкового числа - 3.2.

3.2.3. Оцінка класифікатора

У цьому кроці, на прикладі контрольної вибірки з зображеннями, ми маємо можливість протестувати точність класифікатора.

Функція *evaluate* порівнює прогнозовану букву нейронної мережі з справжньою буквою.

```
def evaluate(outputs: Variable, labels: Variable) -> float:
    """Evaluate neural network outputs against non-one-hotted labels."""
    Y = labels.numpy()
    Yhat = np.argmax(outputs, axis=1)
    return float(np.sum(Yhat == Y))
```

batch_evaluate застосовує попередню функцію до всіх зображень. *batch* - група зображень, збережені в якості одного тензору. Ми збільшуємо загальну кількість(n) зображень, що ми оцінюємо, на кількість зображень у цій групі(batch). *outputs = net ...* - логічний вивід даної групи зображень. if isinstance - конвертація в массив NumPy. І після цього, ми підраховуємо вірно класифіковані зразки за допомогою функції *evaluate*. *score / n* - обраховуємо результат у відсотках.

```
def batch_evaluate(
        net: Net,
        dataloader: torch.utils.data.DataLoader) -> float:
    """Evaluate neural network in batches, if dataset is too large."""
    score = n = 0.0
    for batch in dataloader:
        n += len(batch['image'])
        outputs = net(batch['image'])
        if isinstance(outputs, torch.Tensor):
            outputs = outputs.detach().numpy()
        score += evaluate(outputs, batch['label'][:, 0])
    return score / n
```

Наступним кроком, експортуємо нашу модель у файл ONNX. При експорті йде перевірка експортованої моделі та запускається логічний вивід. Зрештою, ми побачимо, що експортована модель працює чудово. Крім того, експортована модель не суперечить оригінальній моделі PyTorch.

```
trainloader, testloader = get_train_test_loaders(1)

# export to onnx
fname = "signlanguage.onnx"
dummy = torch.randn(1, 1, 28, 28)
torch.onnx.export(net, dummy, fname, input_names=['input'])

# check exported model
model = onnx.load(fname)
onnx.checker.check_model(model)  # check model is well-formed

# create runnable session with exported model
ort_session = ort.InferenceSession(fname)
net = lambda inp: ort_session.run(None, {'input': inp.data.numpy()})[0]

print('=' * 10, 'ONNX', '=' * 10)
train_acc = batch_evaluate(net, trainloader) * 100.
print('Training accuracy: %.1f' % train_acc)
test_acc = batch_evaluate(net, testloader) * 100.
print('Validation accuracy: %.1f' % test_acc)
```

```
========== PyTorch ==========
Training accuracy: 99.9
Validation accuracy: 97.4
========== ONNX ==========
Training accuracy: 99.9
Validation accuracy: 97.4
```

Точність навчання - 99, 9 %. Точність перевірки 97, 4 %. Отже, результат, що ми отримали показує, що модель навчена відмінно, адже показники є дуже високими.

3.2.4. Прив’язка комп’ютерного зору

На даному етапі, ми прив’язуємо камеру до класифікатора. Наступний код відповідає за те, що збирає вхідні дані, отримані з камери, класифікує жест і повідомляє його користувачу.

```
def main():
    # constants
    index_to_letter = list('АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ')
    mean = 0.485 * 255.
    std = 0.229 * 255.
    ort_session = ort.InferenceSession("signlanguage.onnx")
```

Налаштовуємо пряму трансляцію з камери та зробимо так, щоб дані з камери зчитувались у кожен такт.

```
cap = cv2.VideoCapture(0)
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
```

Додамо функцію, що робить знімок по центру, переводить у сірий колір і змінює розмір на 28x28.

```
def center_crop(frame):
    h, w, _ = frame.shape
    start = abs(h - w) // 2
    if h > w:
        frame = frame[start: start + w]
    else:
        frame = frame[:, start: start + h]
    return frame
```

```
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # preprocess data
    frame = center_crop(frame)
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    x = cv2.resize(frame, (28, 28))
    x = (frame - mean) / std
```

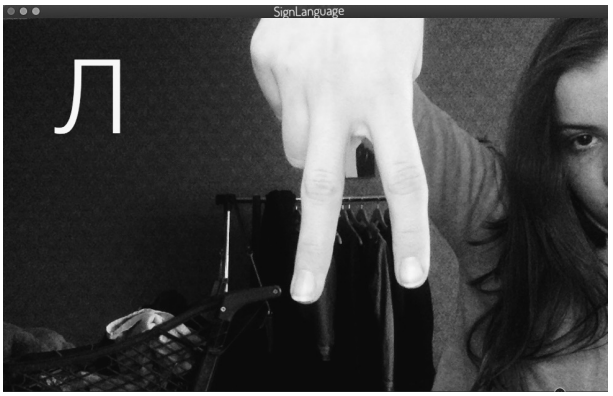
Конвертуємо вивід у букву. Додамо відображення букви на фото.

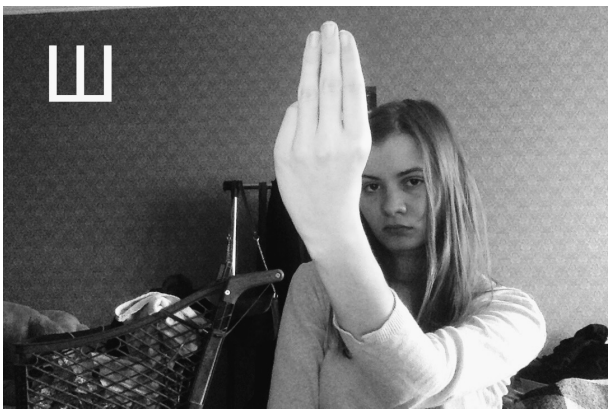
```
x = (frame - mean) / std

x = x.reshape(1, 1, 28, 28).astype(np.float32)
y = ort_session.run(None, {'input': x})[0]

index = np.argmax(y, axis=1)
letter = index_to_letter[int(index)]
```

```
letter = index_to_letter[int(index)]

cv2.putText(frame, letter, (100, 100), cv2.FONT_HERSHEY_SIMPLEX, 2.0, (0, 255, 0), thickness=2)
cv2.imshow("Sign Language Translator", frame)
```

3.3. Фінальний результат

Після ініціалізації кінцевого файлу, відкривається вікно з відео з веб-камери комп’ютеру. У верхньому лівому куті відображається прогнозована літера. Необхідно затримати руку у правильному положенні, у вигляді бажаного жесту. Приклади виконання:





Висновки

Можна зробити висновок, що поставлена ціль була успішно виконана. Під час роботи над проектом було проведено аналіз основних існуючих видів та підходів машинного навчання. Крім того, було проведено огляд існуючих технологій, середовищ розробки та бібліотек. У результаті виконання роботи були розроблені: перекладач по розпізнаванню жестової мови за допомогою комп'ютерного зору та машинного навчання. Також були задіяні такі аспекти навчання моделі, як приріст даних для її надійності та експорт моделей штучного інтелекту за допомогою інструменту ONNX. Остаточним продуктом є застосунок з комп'ютерним зором у режимі роботи в реальному часі, що перекладає жестову мову в букви, за допомогою створеного конвеєру.

У найближчому майбутньому заплановане вдосконалення застосунку: збільшення кількості жестів української мови та розширення бази іншими мовами(англійська, російська і так далі).

Список використаної літератури

1. Статистика по Україні по кількості людей з вадами слуху: https://web.archive.org/web/20190703084103/https://www.ethnologue.com/language/ukl
2. Дані Всесвітньої організації глухонімих: https://www.ethnologue.com/language/ukl
3. Дані по кількості жестових мов у всьому світі: https://www.ukrinform.ua/rubric-culture/2543744-suprun-sprostovue-mif-pro-edinu-movu-zestiv.html
4. Дослідження вчених про те, як виникають мови жестів: https://glavcom.ua/digest/yak-vinikaje-mova-645729.html
5. Перші відкриття шкіл для глухонімих в Україні: https://uain.press/blogs/zhestova-mova-po-ukrayinsky-1083664
6. Supervised learning: Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar (2012) Foundations of Machine Learning
7. Unsupervised learning: Duda, Richard O.; Hart, Peter E.; Stork, David G. (2001). "Unsupervised Learning and Clustering". Pattern classification (2nd ed.)
8. CNN Convolutional Neural Network: https://brilliant.org/wiki/convolutional-neural-network/