МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.
«_____»_____2022 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
«МАГІСТР»

Тема: «Нейромережевий застосунок класифікації статичних і динамічних дактилем жестової мови»

Виконавець: Балицька Ірина Андріївна

Керівник: Глазок Олексій Михалович

Нормоконтролер: Глазок Олексій Михалович

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Литвиненко О.Є.
«_____»_____2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

Балицької Ірини Андріївни
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема кваліфікаційної роботи: Нейромережевий застосунок класифікації статичних і динамічних дактилем жестової мови

затверджена наказом ректора від «16» вересня 2022 р. № 1530/ст

2. Термін виконання роботи: з 05.09.2022 по 30.11.2022

3. Вихідні дані до роботи: завдання на кваліфікаційну роботу, існуючі розробки та наукові роботи, мова програмування *Python*, штучні нейронні мережі, дактилеми, комп'ютерний зір

4. Зміст пояснювальної записки:

1) аналіз предметної області та постановка завдання;

2) проектування застосунку;

3) розробка та тестування застосунку.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1. процес класифікації жесту (схема алгоритму);

2. діаграма класів;

3. екранна форма веб-інтерфейсу користувача;

4. графіки залежностей точності від епохи та втрат від епохи;

5. структура файлів та директорій проєкту.

6. Календарний план-графік

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1	Провести аналіз літератури за темою кваліфікаційної роботи. Провести аналіз існуючих засобів розпізнавання дактилем та мови жестів.	05.09.22-15.09.22	
2	Підготувати набір даних для навчання нейронної мережі	15.09.22-19.09.22	
3	Спроєктувати та навчити нейронну мережу. Розробити модуль розпізнавання.	19.09.22-01.10.22	
4	Підготувати текст першого розділу пояснювальної записки	03.10.22-12.10.22	
5	Підготувати текст другого розділу пояснювальної записки	12.10.22-21.10.22	
6	Розробити додаток та його інтерфейс. Провести тестування додатку. Підготувати текст третього розділу пояснювальної записки	22.10.22-05.11.22	
7	Завершити оформлення пояснювальної записки. Оформити графічний матеріал.	07.11.22-10.11.22	
8	Пройти нормоконтроль. Підготувати доповідь та презентацію до захисту.	10.11.22-30.11.22	

7. Дата видачі завдання: «05» вересня 2022 р.

Керівник кваліфікаційної роботи Глазок О.М.
(підпис керівника)

Завдання прийняв до виконання Балицька І.А.
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Нейромережевий застосунок класифікації статичних і динамічних дактилем жестової мови» містить: 102 с., 73 рис., 43 літературних джерел.

Ключові слова: ШТУЧНА НЕЙРОННА МЕРЕЖА, ДАКТИЛЕМИ, КОМП'ЮТЕРНИЙ ЗІР, *PYTHON, KERAS, OPENCV.*

Об'єкт дослідження – процес класифікації статичних і динамічних жестів, що відповідають дактилемам українського алфавіту мови жестів із використанням нейронної мережі та комп'ютерного зору.

Предмет дослідження – застосунок із використанням нейронної мережі, що реалізує класифікацію дактилем дактильного алфавіту.

Мета виконання кваліфікаційної роботи – реалізувати застосунок для класифікації статичних і динамічних дактилем.

Методи дослідження – порівняльний аналіз фреймворків глибинного навчання, проєктування алгоритму класифікації жесту, написання функцій нейронної мережі та комп'ютерного зору.

Практичне значення – в результаті виконання кваліфікаційної роботи створено прототип застосунку, який може використовуватися як окремо, для перекладу жесту на літеру алфавіту, так і як доповнення до інших систем. Використання застосунку є доречним в освітніх закладах, при навчанні людей з мовно-слуховими дефектами, для комунікації між чуючими та нечуючими людьми.

Галузь застосування – повсякденне життя, заклади освіти.

Прогнозні припущення щодо розвитку об'єкта дослідження – збільшення набору даних, розробка мобільного застосунку.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ДА – дактильний алфавіт

КЗ – комп’ютерний зір

РНМ – рекурентна нейронна мережа

ШНМ – штучна нейронна мережа

LSTM – *Long Short-Term Memory* (довга короткочасна пам'ять)

ВСТУП

Мова жестів є одним із найбільш популярних підходів у спілкуванні між людьми з вадами слуху (як вродженими так і набутими). На сьогодні в Україні за різними даними налічується більш ніж 100 тис. таких осіб. У сучасних реаліях це створює серйозні перешкоди для повноцінної комунікації глухих та слабочуючих людей з їх безпосереднім оточенням та обмежує їх адаптацію і спроби участі в широкому суспільному житті.

Жестовою мовою називають спеціальний вид мовлення, який дає можливість позначати літери, слова та словосполучення окремими комбінаціями долонь та пальців рук. Жестова мова, яку використовують люди з особливими потребами, має свій словник і синтаксис, який суттєво відрізняється від усної/письмової мови. У світі жестомовних осіб існує власний «ручний» алфавіт – дактильний алфавіт, у якому кожна літера відповідає окремій комбінації пальців руки. У такий спосіб «промовляються» імена, географічні назви та слова, які не мають усталеного жестового позначення.

Дактильний алфавіт складається із статичних та динамічних жестів. Статичний жест – це нерухома поза руки, показана людиною у певний момент часу, одночасно з тим динамічний жест являється послідовністю змінних поз руки.

Дослідники активно вивчають методи розробки систем розпізнавання мови жестів в реальному часі. На відміну від задачі розпізнавання статичних жестів, що має ряд якісних практичних рішень, задача розпізнавання динамічних жестів не вирішена у повному обсязі тому є перспективним та актуальним науково-технічним завданням. Через те, що існуючі методи мають ті чи інші функціональні обмеження, дана область вимагає досліджень і розробки нових методів.

Метою даної роботи є створення застосунку класифікації статичних та динамічних дактилем жестової мови. Такий застосунок допоможе людям з вадами слуху краще адаптуватися у суспільстві та дозволить ліпше соціалізуватися з оточуючими.

Для досягнення заявленої мети необхідно виконати наступні завдання:

1) сформувати завдання класифікації дактилем та розглянути існуючі наукові дослідження та методи реалізації;

2) розробити алгоритм роботи програми;

3) написати код нейронної мережі та функцій комп'ютерного зору;

4) здійснити навчання та тестування нейронної мережі;

5) протестувати застосунок та перевірити коректність класифікації;

6) розробити інтерфейс користувача.

Актуальність теми. На сьогоднішній день актуальність даної теми є значною, оскільки застосунків перекладу українських дактилем у відкритому доступі невелика кількість, незважаючи на наявність наукових праць та статей за цією темою. Застосунки класифікації дактилем мови жестів потрібні для полегшення спілкування між людьми з мовно-слуховими вадами та іншими, вони можуть застосовуватися для звичайного перекладу, а також для навчання.

Об'єкт дослідження – процес класифікації статичних і динамічних жестів, що відповідають дактилемам українського алфавіту мови жестів із використанням нейронної мережі та комп'ютерного зору.

Предмет дослідження – застосунок із використанням нейронної мережі, що реалізує класифікацію дактилем дактильного алфавіту.

Методи дослідження – нейронні мережі, функції комп'ютерного зору, метод класифікації жестів.

Кваліфікаційна робота складається з трьох розділів.

Перший розділ присвячений аналізу предметної області та визначенню завдання до кваліфікаційної роботи.

У другому розділі виконується проєктування застосунку.

Третій розділ містить процес розробки застосунку та тестування

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Завдання класифікації дактилем

Мова жестів – це візуально-жестова мовна система, яка використовується як основний засіб комунікації людей з різними вадами слуху. Ця мова не є універсальною. У різних країнах світу жестомовні особи спілкуються різними мовами, які мають свою власну лексико-граматичну структуру сформовану на основі територіально-мовних особливостей.

Всі символи любої жестової мови організовані лінгвістичним способом. Кожен жест складається з трьох окремих частин: форма руки, положення рук і рух рук.

В Україні також існує своя жестова мова. Український дактильний алфавіт (ДА) є допоміжною системою української мови жестів та складається з дактилем – комбінацій пальців руки, що відповідають літерам українського алфавіту (рис. 1.1). ДА використовується для жестового промовляння імен, географічних назв та слів, які не мають усталеного жестового позначення.

Для глухих та слабочуючих людей використання жестової мови, та, зокрема, дактильного алфавіту, є зручною формою спілкування. Однак її вагомим недоліком є обмеженість контактів з іншими людьми, оскільки вивчення мови жестів потребує багато часу та зусиль, і переважна більшість населення країни, не маючи до цього мотивації, цією мовою не володіє.

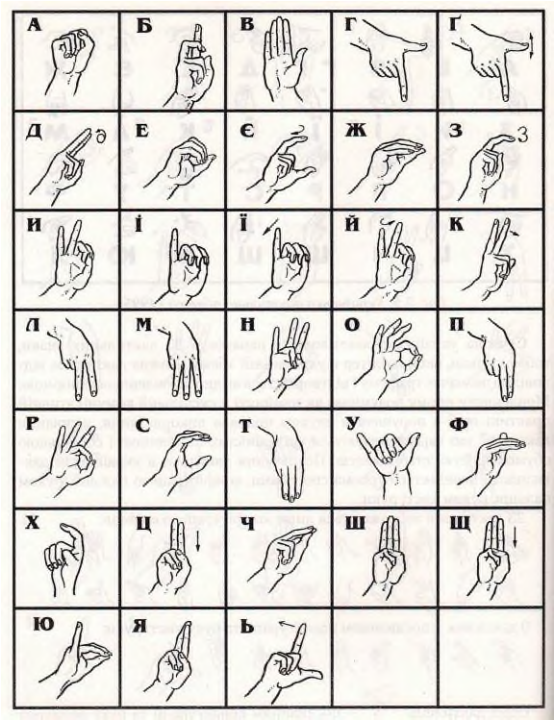


Рис. 1.1. Український дактильний алфавіт

Вирішення цієї проблеми може бути знайдене у світі машинного навчання та комп'ютерного зору. Впровадження технології прогнозної моделі для автоматичної класифікації символів дактильного алфавіту можна використовувати для перекладу дактилем у реальному часі. Таку технологію можна буде застосовувати як окремо – для перекладу, так і в інтеграції з іншими: у віртуальних конференціях у якості субтитрів, у навчальних програмах з метою тренування, тощо.

1.2. Аналіз сучасних розробок розпізнавання і класифікації мови жестів та дактилем

1.2.1. Існуючі застосунки

Станом на листопад 2022 року, у відкритому доступі є не так багато застосунків, що виконують переклад мови жестів або дактилем. Проте, існують програми та сайти, що допомагають вивчати та практикувати жестову мову,

або здійснювати переклад слів та речень на мову жестів. Нижче наведені вітчизняні та зарубіжні розробки за визначеною тематикою.

Комп'ютерний додаток «*Розпізнавання українського дактильного алфавіту*» (рис. 1.2–1.4) розроблений Іриною Балицькою. Файли даного застосунку розміщено у репозиторії веб сервісу контролю версій *GitHub* (*https://github.com/IrinaBalitskaya/Ukrainian-manual-alphabet-recognition*). Працює на операційній системі *Windows*, для використання бажано мати веб-камеру з роздільною здатністю 1280x720 пікселів.

Переваги додатку:

– простий інтерфейс;

– швидкість роботи;

– розпізнавання жестів дактильного алфавіту у режимі реального часу з веб-камери (рис. 1.2);

– розпізнавання жестів з відео (рис. 1.3);

– розпізнавання жестів з папки з зображеннями (рис. 1.4).

Недоліки додатку:

– працює лише на операційній системі *Windows*;

– потрібно завантажувати та встановлювати.

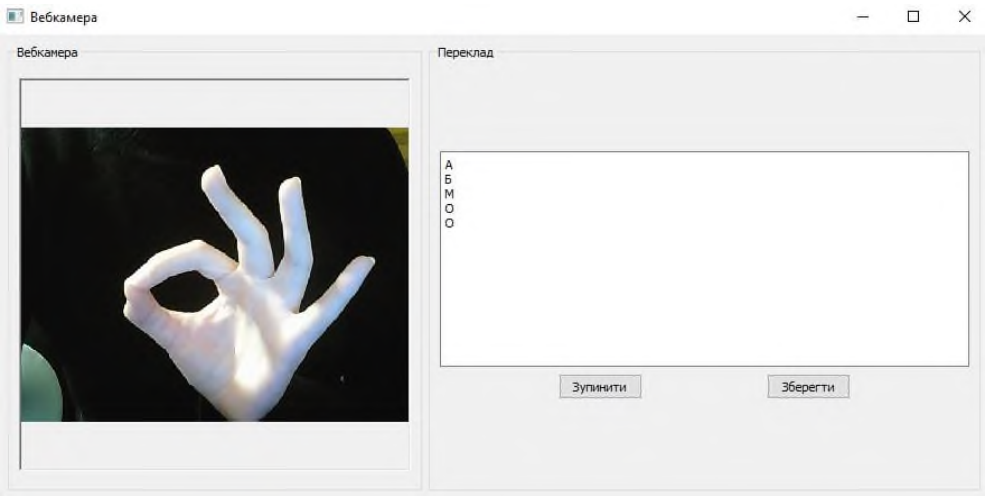


Рис. 1.2. Розпізнавання жестів з веб-камери

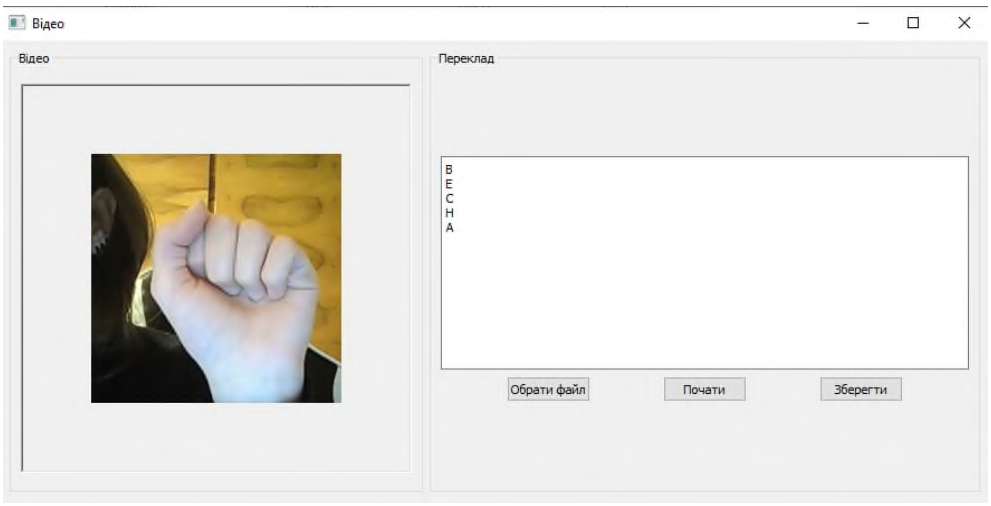


Рис. 1.3. Розпізнавання жестів з відео

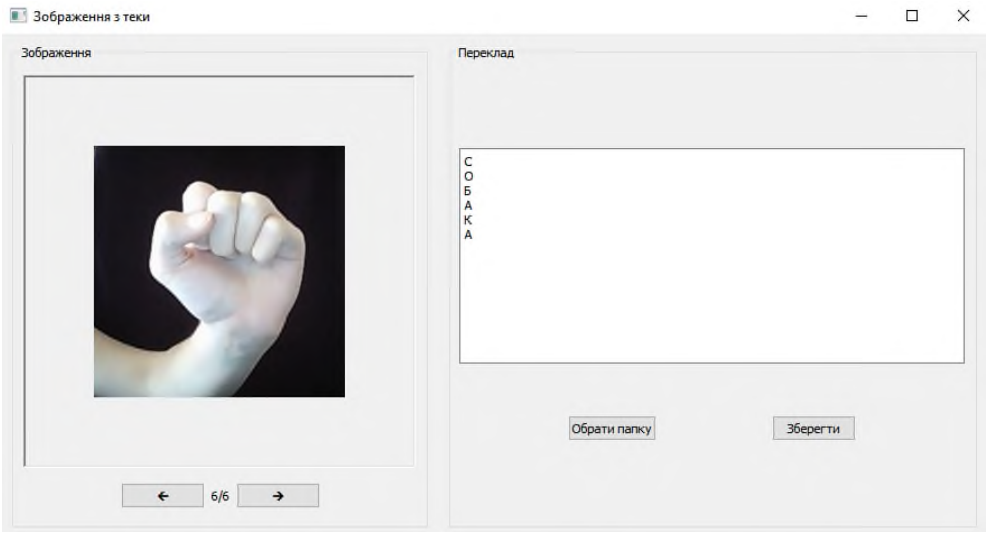


Рис. 1.4. Розпізнавання жестів з папки з зображеннями

Веб-сайт «*Spreadthesign*» (*spreadthesign.com/uk.ua/*) адмініструється міжнародною асоціацією «Європейський центр жестових мов», на якому зібрано та задокументовано понад 400000 жестів. Сайт є перекладачем тексту або літер на мову жестів (рис. 1.5) чи дактильний алфавіт (рис. 1.6) відповідно.

Переваги:

– переклад здійснюється на 43 мови, у тому числі й українську;
– переклад виконаний у вигляді *GIF*-анімацій та простих зображень;
– простий та зрозумілий інтерфейс.

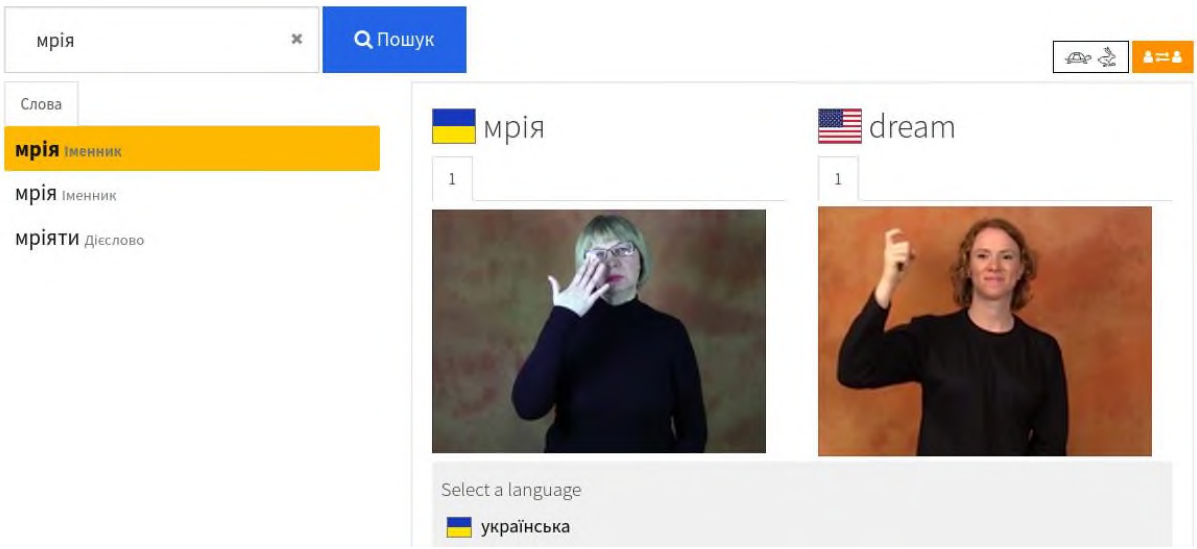


Рис. 1.5. Переклад слів

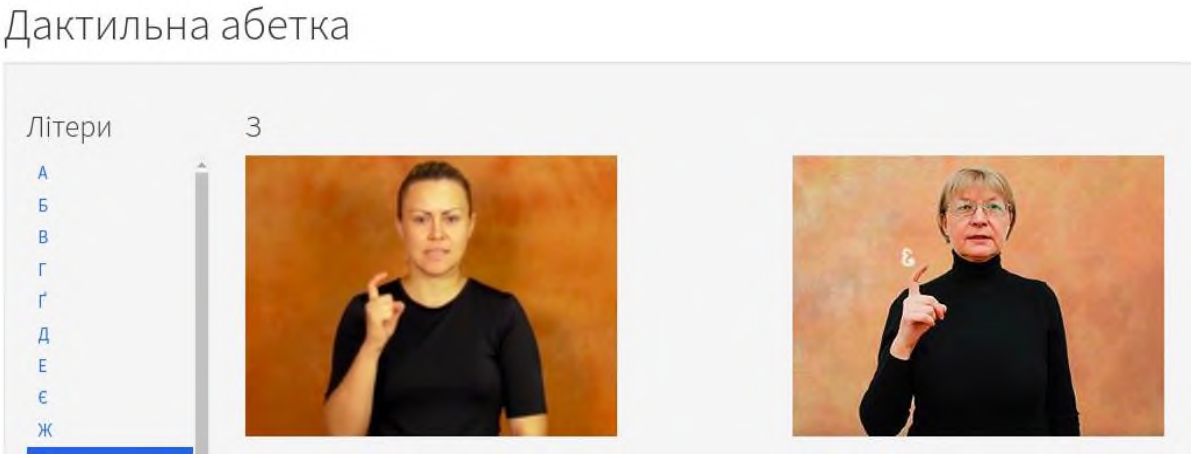


Рис. 1.6. Переклад літер

Мобільний застосунок «*Hand Talk*» (*https://www.handtalk.me/en/app/*) (рис. 1.7) розроблений для перекладу слів, літер або речень (рис. 1.8) на мову жестів.

Переваги:

– підтримує переклад надрукованого тексту та аудіо;

– перекладені жести показує анімований аватар, який можна обертати на 360°;

– можливість регулювати швидкість відображення жесту;

– перекладає літери, слова та речення.

Недоліки:

– підтримує переклад лише з англійської на американську мову жестів та з португальської на бразильську мову жестів.

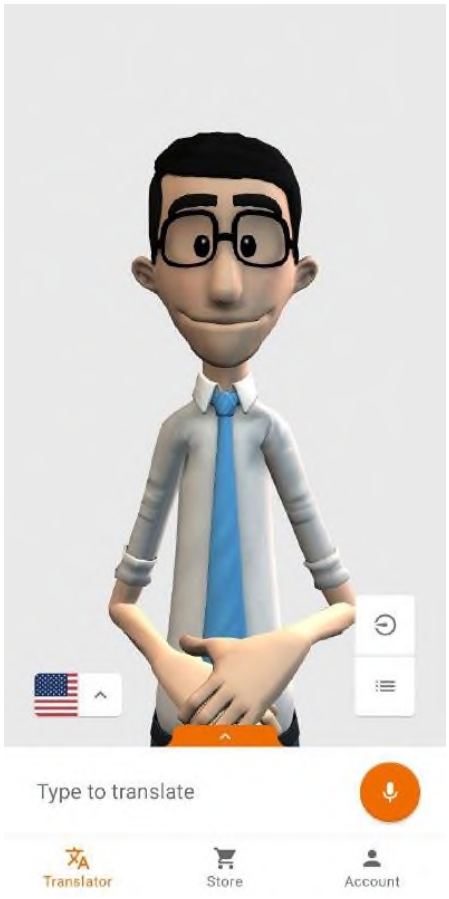


Рис. 1.7. Застосунок *Hand Talk*

Рис. 1.8. Переклад речення

1.2.2. Сучасні наукові дослідження

Проблема Класифікації та розпізнавання жестової мови і дактильного алфавіту привертає увагу багатьох вітчизняних та зарубіжних науковців.

У статті [1] досліджено низку методів розпізнавання українського дактильного алфавіту, наведено концептуальні недоліки кожного із методів, на основі яких запропоновано можливі рішення для удосконалення.

У роботі [2], проаналізовано алгоритм виділення ознак жесту для виявлення кінців пальців при спілкуванні дактильним алфавітом, а також ідентифікації кисті за еталоном.

Методику розпізнавання жестів за кінцями пальців розглянуто ще в одній роботі [3]. Для ідентифікації жесту була створена ШНМ прямого поширення, а метод її навчання був заснований на методі найшвидшого спуску.

Відстеження та розпізнавання жестів з веб-камери за контуром руки репрезентовано у публікації [4]. Навчена нейронна мережа здатна розпізнавати жести незалежно від освітлення в кадрі чи кута, під яким зображується жест.

У статті [5] представлено систему розпізнавання українського дактильного алфавіту за допомогою згорткової нейронної мережі, що була навчена на власноруч створеному датасеті українських дактилем.

Класифікатор американського дактильного алфавіту з використанням сучасної архітектури згорткової ШНМ під назвою *Wide Residual Network*, навченої з процедурою *Snapshot Learning* висвітлено у статті [6]. Нейромережу було навчено на розширених наборах даних, доступних на веб-сторінках Університету Суррея та Університету Мессі, з використанням трансферного навчання.

У праці [7] закцентовано увагу на розпізнавання дактильного алфавіту американської мови жестів на відео з соціальних мереж (в умовах природного спілкування), що не є професійно відзнятими для даної роботи. За словами авторів, дана робота спрямована на вирішення проблеми реальних даних, зменшуючи потребу в модулях виявлення або сегментації, які зазвичай використовуються в цій області. Основним підходом для цього є динамічне фокусування на областях інтересу із високою роздільною здатністю.

У роботі [8] опубліковано дослідження з розпізнавання американського дактильного алфавіту і жестових цифр. При навчанні згорткової нейронної мережі використовуються підмножини даних про глибину. Завдяки різним конфігураціям навчання, наприклад вибір гіперпараметрів із перевіркою та без перевірки, було досягнуто точності розпізнавання в 99% для людей, на чиїх руках навчалася нейронна мережа та до 85.5% для людей, що не були залучені до навчання.

У [9] автором описано систему, що здатна інтерпретувати жести рук з американської мови жестів і перетворювати їх на текст або в усну мову та навпаки.

1.3. Застосування нейронних мереж у класифікації жестів

Нейронні мережі, також відомі як «штучні нейронні мережі» (ШНМ), є одним з методів машинного навчання та основою алгоритмів глибинного навчання. Їх назва та структура навіяні людським мозком. ШНМ імітують спосіб, яким біологічні нейрони передають сигнали один одному.

ШНМ складаються з вузлових шарів: вхідний шар, один або більше прихованих шарів і вихідний шар (рис. 1.9). Кожен вузол, або штучний нейрон, з'єднується з іншим і має відповідну вагу та поріг спрацьовування. Якщо вихідний сигнал будь-якого окремого нейрона перевищує вказане порогове значення, цей нейрон активується, надсилаючи дані на наступний шар мережі; в іншому випадку дані не передаються.

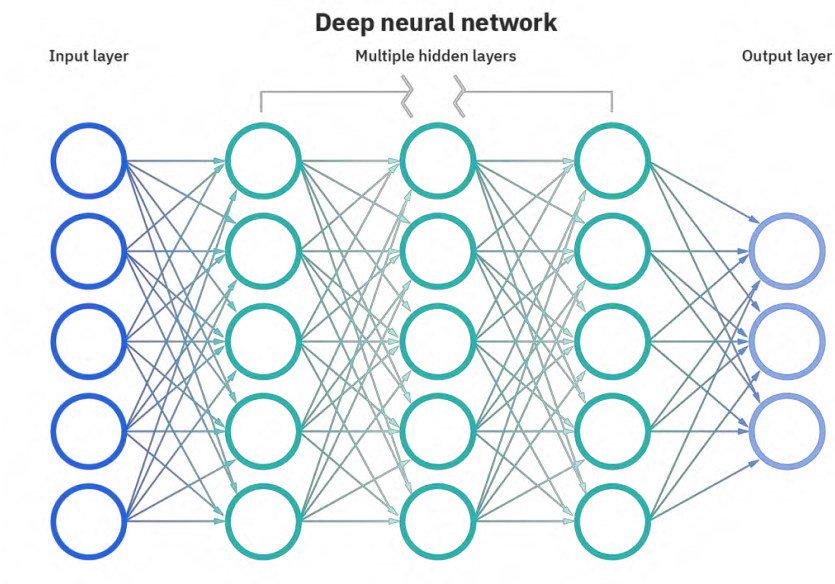


Рис. 1.9. Глибинна нейронна мережа

Розробники ШНМ використовують тренувальні дані, щоб навчати мережу та з часом підвищувати її точність. Після того, як у результаті навчання мережі набули необхідну точність, вони стають потужними інструментами в інформатиці та штучному інтелекті, що дозволяє класифікувати та кластеризувати дані з високою швидкістю. Завдання з розпізнавання мовлення або розпізнавання

зображень можуть виконуватись за хвилини чи години, і це може бути значно швидше, порівняно з ручною ідентифікацією експертами-людьми. [10]

Нейронні мережі можна класифікувати на різні типи, які використовуються для різних цілей.

Персептрон (рис. 1.10) – найстаріша нейронна мережа, створена Френком Розенблатом у 1958 році. Вона складається з одного нейрона і є найпростішою формою ШНМ.

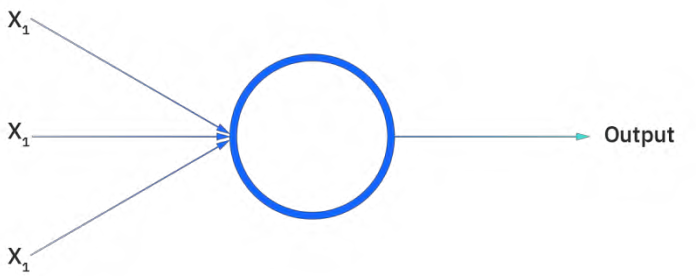


Рис. 1.10. Перцептрон

Нейронні мережі прямого зв'язку, або багатошарові персептрони (БП), складаються з вхідного шару, прихованого шару або декількох шарів і вихідного шару. Хоча ці ШНМ також зазвичай називають БП, важливо зазначити, що насправді вони складаються з сигмовидних нейронів, а не з перцептронів, оскільки більшість проблем реального світу є нелінійними. Дані зазвичай подаються в ці моделі для їх навчання, і вони є основою для комп'ютерного зору, обробки природної мови та інших ШНМ.

Згорткові нейронні мережі схожі на мережі прямого зв'язку, але вони зазвичай використовуються для розпізнавання зображень, розпізнавання образів та/або комп'ютерного зору. Ці мережі використовують принципи лінійної алгебри, зокрема множення матриць, щоб ідентифікувати шаблони в зображенні. [11]

Рекурентна нейронна мережа (РНМ) – це тип штучної нейронної мережі, яка використовує послідовні дані або дані часових рядів. Ці алгоритми глибокого навчання зазвичай використовуються для порядкових або часових проблем, таких як переклад мови, обробка природної мови, розпізнавання мовлення та підписи до

зображень. Подібно до ШНМ прямого зв'язку та згорткових ШНМ, рекурентні ШНМ використовують тренувальні дані для навчання.

Вони відрізняються своєю «пам'яттю», оскільки вони беруть інформацію з попередніх вхідних даних, щоб впливати на поточні вхідні та вихідні дані. Хоча традиційні глибинні нейронні мережі припускають, що входи та виходи незалежні один від одного, вихід рекурентних нейронних мереж залежить від попередніх елементів у послідовності. Майбутні події також були б корисними для визначення результату даної послідовності, проте односпрямовані рекурентні нейронні мережі не можуть врахувати ці події у своїх прогнозах.

Іншою відмінною характеристикою рекурентних мереж є те, що вони спільно використовують параметри на кожному рівні мережі. У той час як мережі прямого зв'язку мають різні ваги для кожного вузла, рекурентні мають однаковий параметр ваги на кожному рівні. Тим не менш, ці ваги все ще коригуються за допомогою процесів зворотного поширення та градієнтного спуску, щоб полегшити навчання з підкріпленням. [12]

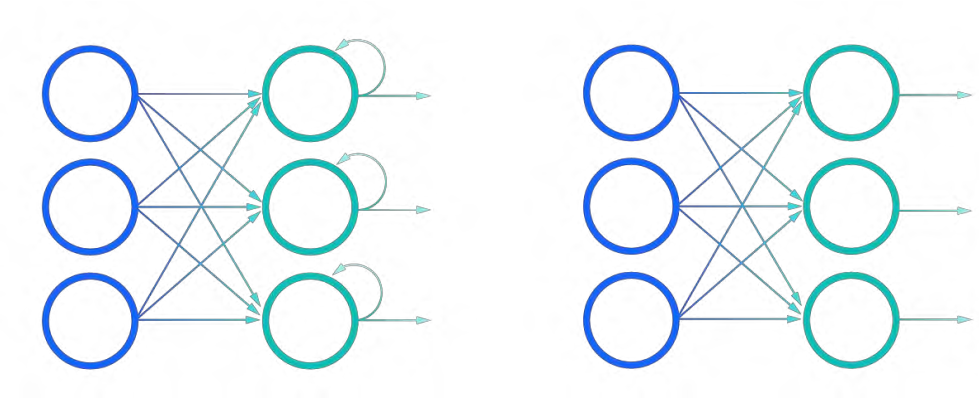


Рис. 1.11. Порівняння рекурентних нейронних мереж (ліворуч) і нейронних мереж прямого зв'язку (праворуч)

Мережі довгострокової короткочасної пам'яті (*LSTM*) – це модифікована версія рекурентних нейромереж, яка полегшує запам'ятовування минулих даних у пам'яті. Її представили Зепп Гохрайтер і Юрген Шмідхубер як рішення проблеми зникнення градієнта. У своїй статті [13] вони працюють над вирішенням проблеми довгострокової залежності.

LSTM переважно використовуються для вивчення, обробки та класифікації послідовних даних, оскільки ці мережі можуть вивчати довгострокові залежності між часовими кроками даних. Загальні застосунки *LSTM* включають аналіз тональності тексту, моделювання мови, розпізнавання мови та аналіз відео. Вона навчає модель за допомогою зворотного поширення. [14]

LSTM дозволяють РНМ запам'ятовувати вхідні дані протягом тривалого періоду часу. Це тому, що *LSTM* містять інформацію в пам’яті, подібно до пам’яті комп’ютера. *LSTM* може читати, записувати та видаляти інформацію.

Цю пам’ять можна розглядати як закриту комірку, що вирішує, зберігати чи видаляти дані, на основі важливості. Присвоєння важливості відбувається через вагові коефіцієнти, які також вивчаються алгоритмом.

У комірці довготривалої короткочасної пам’яті є три типи гейтів («ворота», що пропускають або не пропускають інформацію): вхідний, забутий і вихідний (рис. 1.12). Вони визначають, чи пропускати новий вхід (вхідний гейт), видаляти інформацію, оскільки вона неважлива (гейт забування), чи дозволяти йому впливати на вихід на поточному часовому етапі (вихідний гейт). [15]

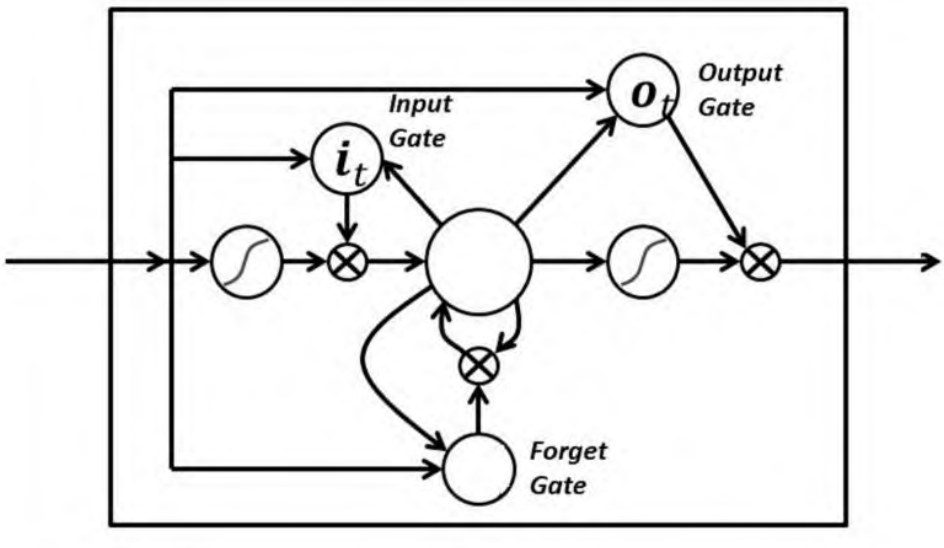


Рис. 1.12. Комірка *LSTM* із трьома типами гейтів

Гейти в *LSTM* є аналоговими у формі сигмовидів, тобто вони варіюються від нуля до одиниці. Той факт, що вони є аналоговими, дозволяє їм здійснювати зворотне поширення.

Проблема зникнення градієнтів вирішується за допомогою *LSTM*, оскільки він підтримує градієнти достатньо крутими, що зберігає відносно коротке навчання та високу точність.

1.4. Застосування комп'ютерного зору у класифікації жестів

Комп'ютерний зір є основною областю технології штучного інтелекту, яка дозволяє комп'ютерам отримувати інформацію з цифрових зображень і відео з метою виконання певних дій. Хоча існують різні визначення штучного інтелекту, усі концепції загалом зосереджені на відтворенні людського інтелекту за допомогою машин.

Технологія машинного зору спрямована на імітацію людського зору, вона передбачає інтерпретацію цифрових зображень або відео з метою розуміння та розпізнавання об'єктів і сцен у них. Це досягається за допомогою поєднання програмного та апаратного забезпечення, яке імітує роботу зорової системи людини. Деякі поширені методи, які використовуються в машинному зорі, це класифікація образів, виділення ознак і обробка зображень.

КЗ включає в себе набір методів обробки зображень, які дозволяють комп'ютерам бачити та розуміти візуальну інформацію. Комп'ютери діють як «мізки», які виконують складні обчислювальні завдання та застосовують складні алгоритми для аналізу зображень або відео, наданих оптичними датчиками або камерами («очі»). Практично будь-яке зображення будь-якої камери можна використовувати для застосування алгоритмів бачення.

Здатність навчити комп'ютери бачити за допомогою штучного інтелекту та сприймати фізичний світ за допомогою візуальних датчиків стає невід'ємною технологією для ефективної оцифровки та автоматизації операцій. В останні роки технології машинного навчання, особливо глибинного навчання,

продемонстрували великий успіх у застосуванні комп'ютерного зору в різних галузях.

Більшість додатків використовують штучний інтелект з Інтернетом речей, хмарні обчислення та *Edge AI*, щоб забезпечити та розгорнути комп'ютерний зір будь-де та в масштабі. Використовуючи технологію зору зі штучним інтелектом, комп'ютери можуть розпізнавати обличчя, читати почерк, розпізнавати об'єкти, класифікувати людські рухи, виконувати автоматичний огляд або автоматично виявляти критичні ситуації за допомогою розпізнавання зображень. [16]

КЗ використовується різними способами:

– Сегментація зображення розбиває зображення на кілька областей або частин, які потрібно досліджувати окремо.

– Розпізнавання об'єктів ідентифікує конкретний об'єкт на зображенні. Ці моделі використовують координати *X*, *Y* для створення обмежувальної рамки та ідентифікації всього, що знаходиться всередині рамки.

– Розпізнавання обличчя – це передовий тип виявлення об'єктів, який не лише розпізнає обличчя людини на зображенні, але й ідентифікує конкретну людину.

– Виявлення країв – це техніка, яка використовується для ідентифікації зовнішнього краю об'єкта чи пейзажу, щоб краще визначити, що є на зображенні.

– Виявлення шаблонів – це процес розпізнавання повторюваних форм, кольорів та інших візуальних індикаторів на зображеннях.

– Класифікація зображень групує зображення в різні категорії.

– Зіставлення ознак – це тип виявлення шаблонів, який зіставляє подібності в зображеннях, щоб допомогти їх класифікувати. [17]

У простих програмах комп'ютерного зору може використовуватися лише один із цих методів, але більш удосконалені способи, як-от комп'ютерне бачення для безпілотних автомобілів, покладаються на кілька методів для досягнення своєї мети.

1.5. Аналіз бібліотек глибинного навчання

Для розробки застосунку була обрана мова програмування *Python 3*.

Найбільш популярними бібліотеками на *Python* є: *PyTorch*, *TensorFlow* та *Keras*.

PyTorch – це бібліотека машинного навчання з відкритим кодом для *Python*, використовується для розробки та навчання моделей глибинного навчання на основі нейронної мережі [18].

TensorFlow – це наскрізна платформа з відкритим кодом для машинного навчання. Вона має комплексну, гнучку екосистему інструментів, бібліотек і ресурсів спільноти, яка дозволяє дослідникам просувати найсучасніші технології машинного навчання, а розробникам – легко створювати й розгортати програми на базі машинного навчання [19].

Keras – це *API* глибинного навчання, написаний на *Python*, який працює на платформі машинного навчання *TensorFlow*. Він був розроблений з упором на можливість швидкого експериментування. Здатність якомога швидше переходити від ідеї до результату є ключем до якісного дослідження [20].

Всі три бібліотеки пов'язані одна з одною, а також мають певні базові відмінності [21].

Рівень API

Keras – це *API* високого рівня, здатний працювати поверх *TensorFlow*, *CNTK* і *Theano*. Він завоював прихильність завдяки простоті використання та нескладного синтаксису, що сприяє швидкому розвитку. *TensorFlow* – це фреймворк, який надає *API* високого та низького рівня. З іншого боку, *Pytorch* – це *API* нижчого рівня, орієнтований на пряму роботу з виразами масиву.

Швидкість

У *Keras* продуктивність порівняно повільніша, тоді як *Tensorflow* і *PyTorch* забезпечують подібний темп, який є швидким і підходить для високої продуктивності.

Архітектура

Keras має просту архітектуру. Вона більш читальна і лаконічна. З іншого боку, *Tensorflow* не дуже простий у використанні. *PyTorch* має більш складну архітектуру і гіршу читаємість в порівнянні з *Keras*.

Налагодження

У *Keras* зазвичай доводиться рідше налагоджувати прості мережі. Але у випадку *Tensorflow* виконати налагодження досить складно. З іншого боку, *Pytorch* має кращі можливості налагодження порівняно з двома іншими.

Набір даних

Keras в основному використовується для невеликих наборів даних, оскільки він порівняно повільніший. З іншого боку, *TensorFlow* і *PyTorch* застосовується для високопродуктивних моделей і великих наборів даних, які потребують швидкого виконання.

Популярність

Зі зростанням попиту в галузі *Data Science* відбулося величезне зростання технології глибинного навчання. Завдяки цьому всі три бібліотеки набули досить великої популярності. *Keras* очолює список, за ним йдуть *TensorFlow* і *PyTorch*. Він став відомим завдяки своїй простоті порівняно з двома іншими.

Отже, можна зробити наступні висновки по використанню наведених бібліотек глибинного навчання:

Перевагами *Keras* є: швидке створення прототипів; невеликих наборів даних; підтримки кількох серверних систем.

Перевагами *TensorFlow* є: великий набір даних; висока продуктивність; функціональність; виявлення об'єктів.

Перевагами *PyTorch* є: гнучкість; коротка тривалість навчання; можливості налагодження [22].

1.6. Висновки до розділу

В даному розділі було розглянуто питання щодо розпізнавання мови жестів, зокрема дактильного алфавіту, та методів їх реалізації. Опрацьовано

публікації вітчизняних та зарубіжних авторів з досліджуваної проблематики, проаналізовано існуючі тематичні застосунки. Виокремлено головні вимоги до розробки заявленого застосунку. Зроблено ряд висновків:

- Автоматизація розпізнавання мови жестів та дактильного алфавіту наразі має велике практичне значення, оскільки вона дозволяє людям з мовно-слуховими вадами забезпечувати безперешкодну соціалізацію в суспільстві.
- *LSTM* – різновид рекурентної нейронної мережі, що здатен навчатися довготривалим залежностям. Мережі *LSTM* спроможні запам’ятовувати інформацію протягом тривалих періодів часу, через що навчання не вимагає багато часу. *LSTM* мережа, при достатній кількості елементів може виконати будь-яке обчислення, тобто є універсальною; також вона добре орієнтована на вирішення проблем класифікації.
- Комп’ютерний зір являється невід’ємною частиною класифікації жестів. Його основною задачею є обробка та аналіз зображення чи кадрів відео, виокремлення необхідної інформації та, разом із навченною ШНМ, класифікація до певної категорії.
- В результаті аналізу переваг та недоліків бібліотек глибинного навчання було обрано бібліотеку *Keras*, оскільки вона може працювати з невеликими за обсягом наборами даних та є відносно простою у застосуванні.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ЗАСТОСУНКУ

2.1. Інструменти реалізації

Python 3

Python – це інтерпретована об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Її високорівневі вбудовані структури даних у поєднанні з динамічною типізацією та динамічною прив'язкою роблять її дуже привабливою для швидкої розробки додатків, а також для використання як мови сценаріїв або зв'язувальної мови для з'єднання існуючих компонентів. Простий і легкий для вивчення синтаксис *Python* наголошує на зручності читання і, отже, знижує вартість обслуговування програми. *Python* підтримує модулі та пакети, що сприяє модульності програм та повторному використанню коду. Інтерпретатор *Python* та широка стандартна бібліотека доступні у вихідному або бінарному вигляді безкоштовно для всіх основних платформ і можуть вільно розповсюджуватися.

Часто програмісти закохуються в *Python* через підвищену продуктивність, яку він забезпечує. Оскільки етап компіляції відсутній, цикл редагування–тестування–налагодження виконується неймовірно швидко. Налагоджувати програми на *Python* легко: помилка або неправильне введення ніколи не викликають помилку сегментації. Натомість, коли інтерпретатор виявляє помилку, він створює виняток. Коли програма не перехоплює виняток, інтерпретатор друкує трасування стека.

Відладчик рівня вихідного коду дозволяє перевіряти локальні та глобальні змінні, обчислювати довільні вирази, встановлювати точки зупинки, виконувати код рядково і т.д. Відладчик написаний на самому *Python*, що свідчить про інтроспективну потужність *Python*. З іншого боку, найчастіше найшвидшим способом налагодження програми є додавання у вихідний код кількох операторів друку: швидкий цикл редагування–тестування–налагодження робить цей простий підхід дуже ефективним. [23]

PyCharm

PyCharm – це спеціальне інтегроване середовище розробки *Python* (*IDE*), яке надає широкий спектр основних інструментів для розробників *Python*, тісно інтегрованих для створення зручного середовища для продуктивної розробки на *Python, web* та *Data science*.

Він підтримує дві версії *Python: v2.x i v3.x*.

PyCharm можна запустити на *Windows, Linux* або *Mac OS*. Крім того, він містить модулі та пакети, які допомагають програмістам розробляти програмне забезпечення за допомогою *Python* за менший час і з мінімальними зусиллями. Крім того, його також можна налаштувати відповідно до вимог розробників. [24]

Keras & TensorFlow 2

Keras – це *API* глибинного навчання, написаний на *Python*, який працює на платформі машинного навчання *TensorFlow*. Він був розроблений з упором на можливість швидкого експериментування. Здатність якомога швидше переходити від ідеї до результату є ключем до якісного дослідження.

Keras – це:

– Простий, але не спрощений. *Keras* зменшує когнітивне навантаження розробника, щоб ви зосередилися на тих частинах проблеми, які дійсно важливі.

– Гнучкість – *Keras* використовує принцип поступового розкриття складності: прості робочі процеси мають бути швидкими та легкими, тоді як будь-які складні робочі процеси мають бути можливими через чіткий шлях, який спирається на те, що ви вже навчилися.

– Потужність – *Keras* забезпечує продуктивність і масштабованість у галузі: він використовується організаціями та компаніями, зокрема *NASA, YouTube* або *Waymo*.

TensorFlow 2 – це наскрізна платформа машинного навчання з відкритим кодом. Це можна розглядати як рівень інфраструктури для диференційованого програмування. Він поєднує в собі чотири ключові можливості:

– Ефективне виконання тензорних операцій низького рівня на CPU, GPU або TPU.

– Обчислення градієнта довільних диференційованих виразів.

– Масштабування обчислень для багатьох пристроїв, таких як кластери із сотень графічних процесорів.

– Експорт програм («графіків») у зовнішні середовища виконання, такі як сервери, браузери, мобільні та вбудовані пристрої.

Keras – це високорівневий *API TensorFlow 2*: доступний, високопродуктивний інтерфейс для вирішення проблем машинного навчання з акцентом на сучасне глибоке навчання. Він надає основні абстракції та будівельні блоки для розробки та доставки рішень машинного навчання з високою швидкістю ітерації.

Keras дає інженерам і дослідникам можливість повною мірою використовувати переваги масштабованості та крос-платформних можливостей *TensorFlow 2*: ви можете запускати *Keras* на *TPU* або на великих кластерах *GPU*, а також ви можете експортувати свої моделі *Keras* для запуску в браузері чи на мобільному пристрої. пристрій. [20]

OpenCV

OpenCV (*Open Source Computer Vision Library*) – бібліотека комп’ютерного зору та машинного навчання з відкритим кодом. *OpenCV* було створено, щоб забезпечити загальну інфраструктуру для програм комп’ютерного зору та прискорити використання машинного сприйняття в комерційних продуктах.

Бібліотека містить понад 2500 оптимізованих алгоритмів, що включає повний набір як класичних, так і найсучасніших алгоритмів комп’ютерного зору

та машинного навчання. Ці алгоритми можна використовувати для виявлення та розпізнавання облич, ідентифікації об’єктів, класифікації дій людей у відео, відстеження рухів камери, відстеження рухомих об’єктів, вилучення *3D*-моделей об’єктів, створення *3D*-хмар точок із стереокамер, з’єднання зображень для отримання високої роздільної здатності. зображення цілої сцени, знаходити подібні зображення в базі даних зображень, видаляти червоні очі із зображень, зроблених за допомогою спалаху, стежити за рухами очей, розпізнавати пейзаж і встановлювати маркери для накладання на нього доповненої реальності тощо. *OpenCV* має понад 47 тисяч користувачів спільнота та оціночна кількість завантажень перевищує 18 мільйонів. Бібліотека широко використовується компаніями, дослідницькими групами та державними органами. [25]

MediaPipe

MediaPipe – це фреймворк з відкритим вихідним кодом, представлений *Google*, який допомагає створювати мультимодальні конвеєри машинного навчання. Розробник може створити прототип, не заглиблюючись у написання алгоритмів та моделей машинного навчання, використовуючи наявні компоненти. Ця структура може використовуватися для різних програм для обробки зображень та мультимедіа (особливо у віртуальній реальності), таких як виявлення об'єктів, розпізнавання облич, відстеження рук, відстеження кількох рук та сегментація волосся. *MediaPipe* підтримує різні апаратні та операційні платформи, такі як *Android, iOS* та *Linux*, пропонуючи *API* на *C++, Java, Objective-c* тощо. Це середовище також здатне використовувати ресурси *GPU*.

Mediapipe поставляється з готовими до використання моделями, де розробники можуть почати використовувати його безпосередньо або зі своїми модифікаціями. Виявлення заперечень може бути опрацьовано дуже легко, не споживаючи багато системних ресурсів. Виявлення об'єктів на основі *ML* з камери прямої трансляції з частотою кадрів 30 кадрів на секунду зазвичай потребує великих ресурсів і неможливе через тривалий час виведення. *MediaPipe* досягає цього шляхом паралельного запуску, відстеження та виявлення, тому кожен процес ніколи не буде заблокований іншим. [26]

Numpy

NumPy – це основний пакет для наукових обчислень на *Python*. Це бібліотека *Python*, яка надає об’єкт багатовимірного масиву, різноманітні похідні об’єкти (такі як замасковані масиви та матриці), а також ряд процедур для швидких операцій над масивами, включаючи математичні, логічні, маніпуляції формою, сортування, вибір, введення/виведення. , дискретне перетворення Фур'є, базова лінійна алгебра, основні статистичні операції, випадкове моделювання та багато іншого. [27]

Flask

Flask – це мікровеб-фреймворк, написаний на *Python*. Його класифікують як мікрофреймворк, оскільки він не потребує спеціальних інструментів чи бібліотек. У ньому немає рівня абстракції бази даних, перевірки форми або будь-яких інших компонентів, де вже існуючі бібліотеки сторонніх розробників забезпечують загальні функції.

Flask дуже пітоністичний. Розпочати роботу з *Flask* легко, оскільки для нього не потрібно довго вчитися.

Крім того, він дуже явний, що покращує читабельність. [28]

HTML

HTML (*HyperText Markup Language*) – це стандартна мова розмітки для створення веб-сторінок. Вона дозволяє створювати та структурувати розділи, абзаци та посилання за допомогою елементів *HTML* (блоків веб-сторінки), таких як теги та атрибути. [29]

CSS

CSS означає мову каскадних таблиць стилів і використовується для стилізації елементів, написаних на мові розмітки, наприклад *HTML*. Він відокремлює вміст від візуального представлення сайту. Зв’язок між *HTML* і *CSS* тісно пов’язаний, оскільки *HTML* – це сама основа сайту, а *CSS* – це вся естетика всього веб-сайту. [30]

jQuery

jQuery – це швидкий, невеликий та багатофункціональний фреймворк *JavaScript*. Він є кросплатформним і підтримує різні типи браузерів. *jQuery* також називають «пишіть менше, робіть більше», тому що він займає багато типових завдань, для виконання яких потрібно багато рядків коду *JavaScript*, і зв'язує їх у методи, які можна викликати за допомогою одного рядка коду, коли це необхідно. Фреймворк значно спрощує такі речі, як обхід *HTML*-документів і маніпуляції з ними, обробка подій, анімація та *AJAX* завдяки простому у використанні *API*, який працює в багатьох браузерах. [31]

AJAX

AJAX розшифровується як «асинхронний *JavaScript* і *XML»*. *JavaScript* містить функції надсилання асинхронного http-запиту за допомогою об'єкта *XMLHttpRequest*. *AJAX* передбачає використання цієї здатності *JavaScript* для надсилання асинхронного *http*-запиту та отримання даних *xml* як відповідь (також в інших форматах) і оновлення частини веб-сторінки (за допомогою *JavaScript*) без перезавантаження чи оновлення всієї веб-сторінки. [32]

AWS

Amazon Web Services (*AWS*) – це найпоширеніша у світі хмарна платформа з найширшими можливостями, що надає понад 200 повнофункціональних сервісів для центрів обробки даних по всій планеті. Мільйони клієнтів, у тому числі стартапи, які стали лідерами за швидкістю зростання, найбільші корпорації та передові урядові установи, використовують *AWS* для зниження витрат, підвищення гнучкості та прискореного впровадження інновацій.

AWS надає незрівнянно більше сервісів та їх функцій, ніж будь-який інший постачальник хмарних послуг: від інфраструктурних технологій, таких як інструменти для обчислення, сховища та бази даних, до інновацій, наприклад машинного навчання та штучного інтелекту, озер даних та аналітики, а також Інтернету речей. З ними клієнт зможе швидше, легше та дешевше перенести поточні додатки в хмару та реалізовувати у ньому будь-які можливі проекти.

AWS також надає найширші функціональні можливості для своїх сервісів. Наприклад, *AWS* пропонує на вибір багато баз даних, спеціально створених для різних типів програм, щоб клієнт міг підібрати правильний інструмент для ефективної та економної роботи. [33]

AWS Elastic Beanstalk

Elastic Beanstalk надає можливість швидко розгортати програмне забезпечення та керувати ними в *AWS Cloud*. *Elastic Beanstalk* підтримує програми, розроблені на Go, Java, .NET, Node.js, PHP, Python і Ruby. При розгортанні ПЗ, *Elastic Beanstalk* створює вибрану підтримувану версію платформи та надає один або кілька ресурсів *AWS*, наприклад екземпляри *Amazon EC2*, для його запуску.

Щоб використовувати *Elastic Beanstalk*, потрібно створити програму та завантажити її версію у формі вихідного коду. *Elastic Beanstalk* автоматично запускає середовище, створює та налаштовує ресурси *AWS*, необхідні для запуску коду. Після запуску середовища можливе керування ним і розгортання нових версій програми. [34]

AWS CodePipeline

AWS CodePipeline – це служба безперервної доставки, яка дає змогу моделювати, візуалізувати та автоматизувати кроки, необхідні для випуску програмного забезпечення. За допомогою *AWS CodePipeline* моделюється повний процес випуску для створення свого коду, розгортання в середовищі попереднього виробництва, тестування програми та випуску її для виробництва. Потім *AWS CodePipeline* створює, тестує та розгортає програму відповідно до визначеного робочого процесу щоразу, коли код змінюється. [35]

2.2. Подання даних

Бібліотека *Mediapipe* містить навчену ШНМ для розпізнавання рук. Долоня руки складається з суглобів, які можна подати точками, сполученими відрізками (рис. 2.1).

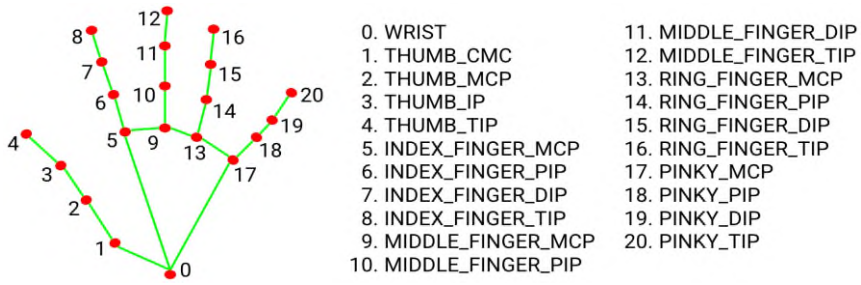


Рис. 2.1. Точки руки у бібліотеці *Mediapipe*

Для лівої та правої руки можна вилучити тривимірний масив координат для кожної точки – *x*, *y* та *z*. *x* та *y* нормалізуються до діапазона [0.0, 1.0] за шириною та висотою зображення відповідно. *z* позначає глибину точки, де глибина зап'ястя є початковою точкою, і чим менше значення, тим ближче точка до камери. Для величини z використовується приблизно той самий масштаб, що й для *x*. [36]

При спробі вилучити координати точок руки, що відсутня на оброблюваному зображенні, *Mediapipe* поверне помилку. Тому, для уникнення цього випадку, координати точки будуть заповнені нулями.

Даний тривимірний масив буде стиснено до одновимірного для полегшення обробки координат нейронною мережею.

Оскільки в дактильному алфавіті присутні динамічні дактилеми, класифікація жестів буде виконуватися не з одного зображення, а з їх послідовності. На відображення одного жесту виділено 30 кадрів (кадрова частота *OpenCV* 30 кадрів в секунду).

Таким чином, жест можна представити двовимірним масивом, що складається з 30 кадрів кожен з яких містить 63 координати точок рук (21 точка помножена на 3 виміри).

2.3. Структура застосунку

Код застосунку складається з 3-х основних модулів: модуль створення датасету, модуль нейронної мережі та модуль класифікації дактилем.

Користувачу застосунку будуть доступні лише функції модуля класифікації дактилем, в якому використовується навчена на створеному датасеті нейронна мережа.

Графічно зв’язок модулів застосунку можна подати наступним чином (рис. 2.2):

Рис. 2.2. Схема зв’язку модулів застосунку

2.4. Проєктування нейронної мережі

2.4.1. Архітектура нейронної мережі

Архітектура нейронної мережі буде використовувати *Sequential* (послідовну) модель та буде складатися з *LSTM* та *Dense* (повнозв'язних) шарів.

Модель – це зазвичай граф шарів. Послідовна модель підходить для простого стеку шарів, де кожен шар має точно один вхідний тензор і один вихідний тензор. [37]

LSTM – шар довгої короткочасної пам'яті. *LSTM* є різновидом архітектури рекурентної нейронної мережі (РНМ). Замість простої рекурсії, він також має «ворота», які регулюють потік інформації через пристрій (рис. 2.3). *LSTM* спочатку були представлені для вирішення проблеми зникаючого градієнта РНМ. Вони часто використовуються замість традиційних, «простих» рекурентних нейронних мереж, оскільки вони також більш ефективні з точки зору обчислень.

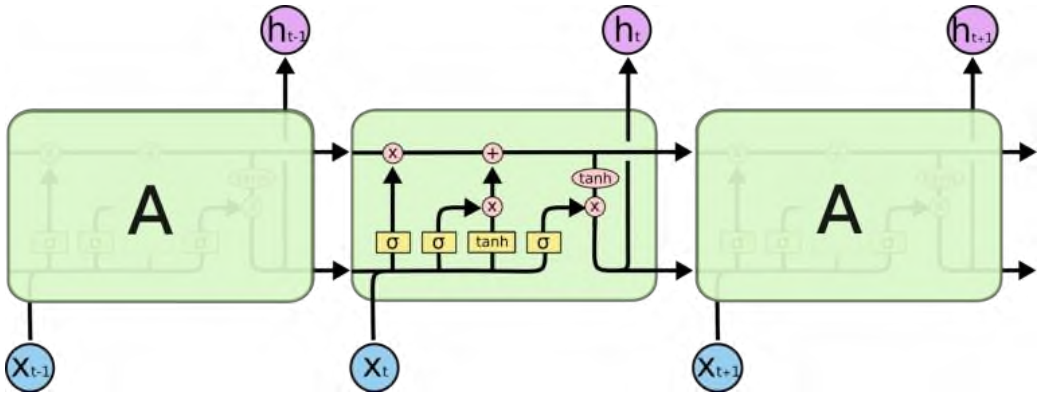


Рис. 2.3. Комірка довгої короткочасної пам'яті (*LSTM*).

Базуючись на доступному апаратному забезпеченні середовища виконання та обмеженнях, цей *LSTM* шар вибере різні реалізації (на основі *cuDNN* або чистого *TensorFlow*), щоб максимізувати продуктивність. Якщо *GPU* доступний і всі аргументи рівня відповідають вимогам ядра *cuDNN*, рівень використовуватиме швидку реалізацію *cuDNN*. [38]

У будь-якій ШНМ повнозв’язний шар – це шар, який глибоко пов’язаний із попереднім шаром, що означає, що нейрони шару пов’язані з кожним нейроном попереднього шару. Цей шар є найбільш часто використовуваним у штучних нейронних мережах. [39]

Нейрон повнозв’язного шару в моделі отримує вихідні дані від кожного нейрона свого попереднього шару, де нейрони повнозв’язного шару виконують множення матриці-вектора.

Як правило, усі шари в послідовній моделі повинні знати форму своїх вхідних даних, щоб мати можливість створити свої ваги. Форму вхідних даних обов’язково потрібно вказати на вході першого шару, оскільки вагові коефіцієнти залежать від форми вхідних даних.

Першим шаром нейронної мережі буде *LSTM* шар. Він складається з 128 нейронів, має функцію активації «*ReLU*», вхідна форма даних шару – 30x63.

Другий шар нейронної мережі – *LSTM*. Складається з 256 нейронів має функцію активації «*ReLU*».

Третій шар – *LSTM*. Складається з 128 нейронів має функцію активації «*ReLU*».

Четвертий шар – повнозв’язний. Складається з 128 нейронів має функцію активації «*ReLU*».

П’ятий шар – повнозв’язний. Складається з 64 нейронів має функцію активації «*ReLU*».

Шостий шар (останній) – повнозв’язний. Складається з 34 (кількість класів) нейронів має функцію активації «*Softmax*».

2.4.2. Активація нейронів

Функції активації – це функції, які використовуються для отримання результату нейрона. Вони використовуються для визначення виходу нейронної мережі, відображають отримані значення в діапазоні від 0 до 1 або від -1 до 1 тощо (залежно від функції).

Функція активації *ReLU* (*Rectified Linear Unit*)

ReLU вважається найбільш використовуваною функцією активації на сьогоднішній день, оскільки вона використовується майже у всіх згорткових нейронних мережах або глибокому навчанні.

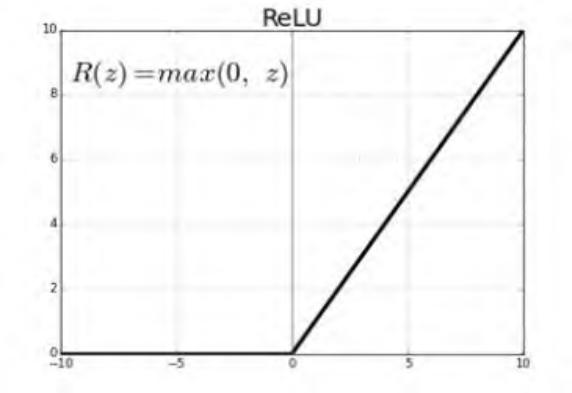


Рис. 2.4. Графік функції активації *ReLU*

ReLU наполовину випрямлена (знизу). *f(z)* дорівнює нулю, коли *z* менше нуля, і *f(z)* дорівнює z, якщо *z* більше або дорівнює нулю.

Діапазон: [від 0 до нескінченності)

І функція, і її похідна монотонні.

Проблема полягає в тому, що при застосуванні такої функції активації всі від'ємні значення відразу стають нульовими, що зменшує здатність моделі належним чином навчатися на основі даних. Це означає, що будь-який негативний вхід, наданий функції активації *ReLU*, одразу перетворює значення на нуль, що, у свою чергу, впливає на кінцевий графік, не відображаючи від'ємні значення належним чином. [40]

Softmax

Softmax часто використовується як остання функція активації ШНМ для нормалізації виходу мережі до розподілу ймовірностей за прогнозованими класами виходу.

Softmax – це функція активації, яка масштабує числа у ймовірності. Результатом *Softmax* є вектор (скажімо, *v*) з імовірностями кожного можливого результату. Сума ймовірностей у векторі *v* дорівнює одиниці для всіх можливих результатів або класів. [41]

2.4.3. Оптимізатор

Оптимізатори – це алгоритми або методи, які використовуються для зміни атрибутів нейронної мережі, таких як ваги та швидкість навчання, щоб зменшити втрати.

Для даної нейронної мережі буде використовуватися оптимізатор Адам.

Оптимізація Адама – це метод стохастичного градієнтного спуску, який базується на адаптивній оцінці моментів першого та другого порядку. [42]

Згідно з *Kingma* та ін., 2014 [43], метод є «обчислювально ефективним, має невеликі вимоги до пам’яті, інваріантний до діагонального масштабування градієнтів і добре підходить для проблем, які є великими з точки зору даних/параметрів».

2.4.4. Функція втрат

Функція втрат – це функція, яка порівнює цільове та прогнозоване вихідні значення; вимірює, наскільки добре нейронна мережа моделює навчальні дані. Під час навчання мета полягає в тому, щоб мінімізувати цю втрату між прогнозованим і цільовим результатами.

Для даної ШНМ буде використовуватися функція втрат *Categorical Cross-Entropy Loss* (категоріальна перехресна ентропія або логарифмічна функція втрат). Категоріальна перехресна ентропія добре підходить для задач класифікації, оскільки один приклад можна вважати належним до певного класу з ймовірністю 1, а до інших категорій з ймовірністю 0.

Категоріальна перехресна ентропія визначається наступним чином:

де s – вектор вихідних значень;

C – кількість класів.

2.5. Проєктування процесу створення набору даних для навчання нейронної мережі

2.5.1. Опис набору даних

Для створення нейронної мережі з високим показником точності потребується чимала кількість навчальних даних. Наразі розроблено багато наборів даних (датасетів) для різних задач. Проте, датасетів з українськими дактилемами, що відповідають даним, представленим вище – немає, тому набір було створено власноруч.

Кількість дактилем в українському дактильному алфавіті відповідає кількості літер у звичайному українському алфавіті. В набір даних увійде 33 жести, а усього датасет буде складатися з 34 класів – додатковим є пустий клас, у якому відсутні жести. Він потрібен для більш коректної класифікації дактилем.

В кожному класі присутньо 60 масивів з координатами, тобто всього в датасеті будет 2040 масивів.

2.5.2. Процес створення набору даних

Для збору датасету застосовується бібліотека комп'ютерного зору *OpenCV*. Вона призначена для обробки, запису та виводу зображень, відео та трансляцій з веб-камери.

Створений датасет збережено у папку з назвою *Dactyls*. В цій директорії присутні 34 папки – класи набору даних. В кожній папці – ще 60 папок – жести. В кожній папці – по 30 файлів з розширенням *.npy* – масиви координат точок руки одного кадру. Таким чином, маємо наступну структуру директорії *Dactyls* (рис. 2.5):

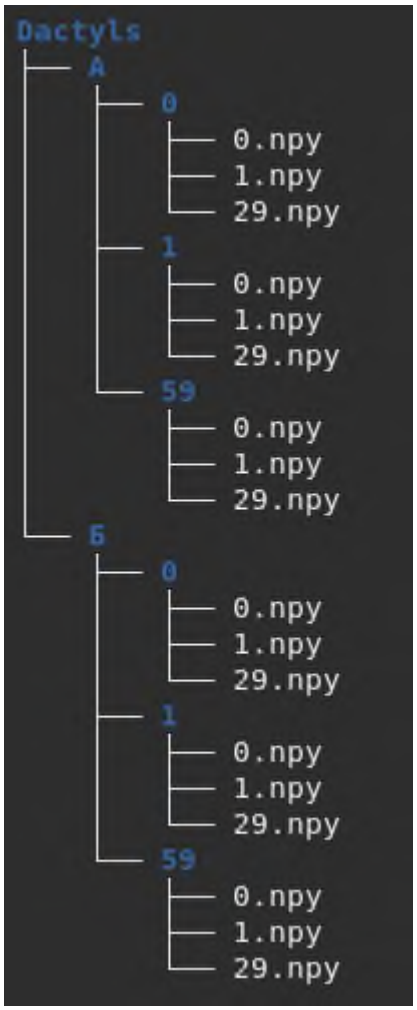


Рис. 2.5. Структура папок та файлів директорії *Dactyls*

Процес збору датасету відбувається наступним чином: запускається модуль датасету, на екрані з'являється вікно з трансляцією з веб-камери (рис.

2.6.). Зверху у вікні написана літера, для якої візуалізується жест, а також послідовний номер жесту. Зібравши 30 кадрів, трансляція зупиняється на 3 секунди, щоб встигнути змінити жест.

Бібліотека *Mediapipe* містить функцію відмальовки точок руки, що з'єднані між собою. Такий скелет буде накладено та відображено на руці під час створення набору даних.

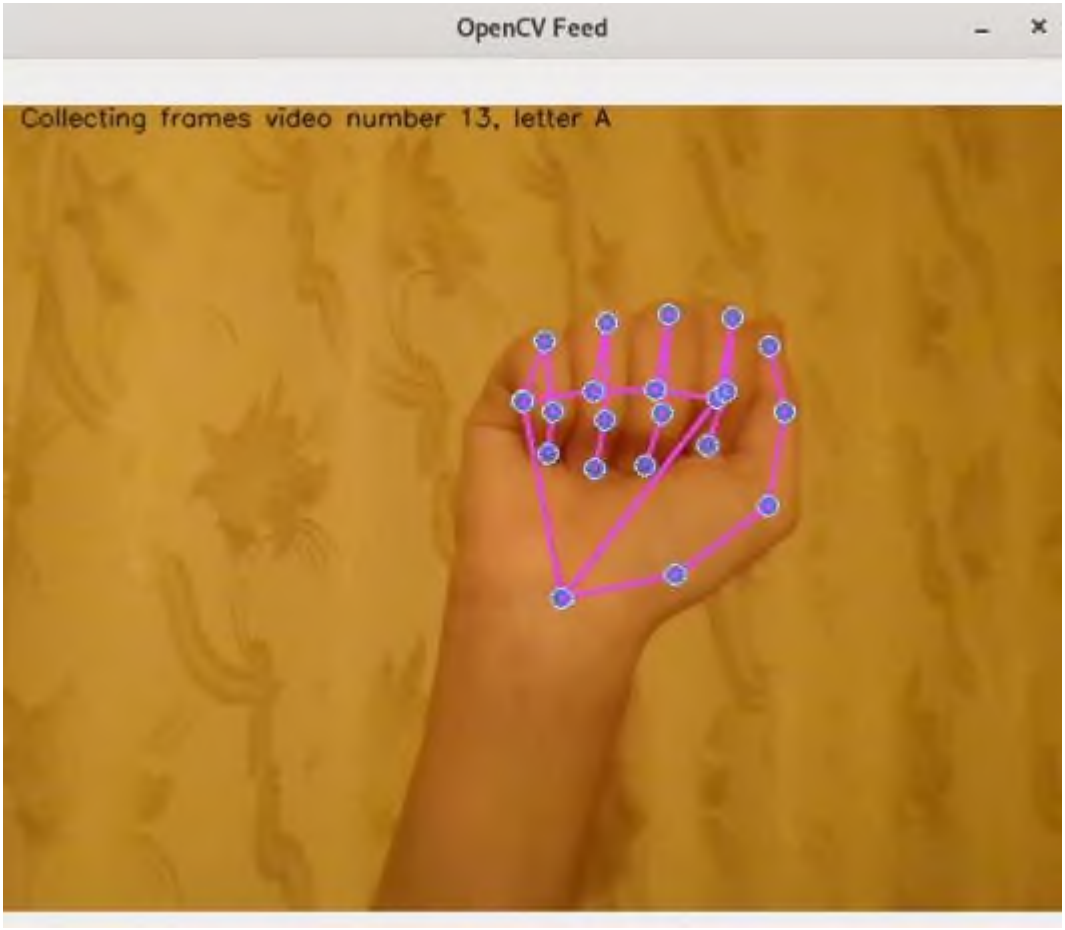


Рис. 2.6. Створення датасету з веб-камери

Після збору даних вони завантажуються з папок та перетворюються в один тривимірний масив 2040x30x63 – 2040 жестів, кожен з яких представлений 30 кадрами, в кожному з якого 63 координати точок рук.

Також створюється одновимірний масив з мітками класу розміром 2040, мітки якого відповідають жестам з попереднього масиву. Кожна мітка – порядковий номер класу жесту (від 0 до 33). Цей масив згодом перетворюється на матрицю, яка має двійкові значення та стовпці, що дорівнюють кількості

категорій у даних. Отримуємо масив 2040x34, що виглядає наступним чином (рис. 2.7.):

```
[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]
[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]
[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]
[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]
[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 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 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 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 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 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 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 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 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 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 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 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 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]
```

Рис. 2.7. Масив міток класів

На останньому етапі датасет розбивається на тренувальні та тестувальні дані. Процент розбиття – 20%, тобто до навчального набору увійде 1632 жести, а до тестувального – 408.

2.6. Проєктування процесу класифікації жестів

Як тільки користувач починає показувати жести рук на веб-камеру, запускається процес класифікації. Перервати його можна лише натиснувши відповідну кнопку. З кожного кадру витягуються координати точок рук, коли збирається 30 масивів координат (через 1 секунду) вони передаються на нейронну мережу. ШНМ повертає вірогідності класів, з яких береться клас з найбільшим відсотком. Якщо відсоток передбачуваного класу більший за 95%, а також якщо

результатом не є клас *null*, переклад дактилеми виводиться користувачу на екран. Після чого процес починається знову.

Графічно процес класифікації дактилем зображений на рис. 2.8.

Рис. 2.8. Процес класифікації дактилем

Алгоритм відображення перекладу дактилеми наведено на рис. 2.9.

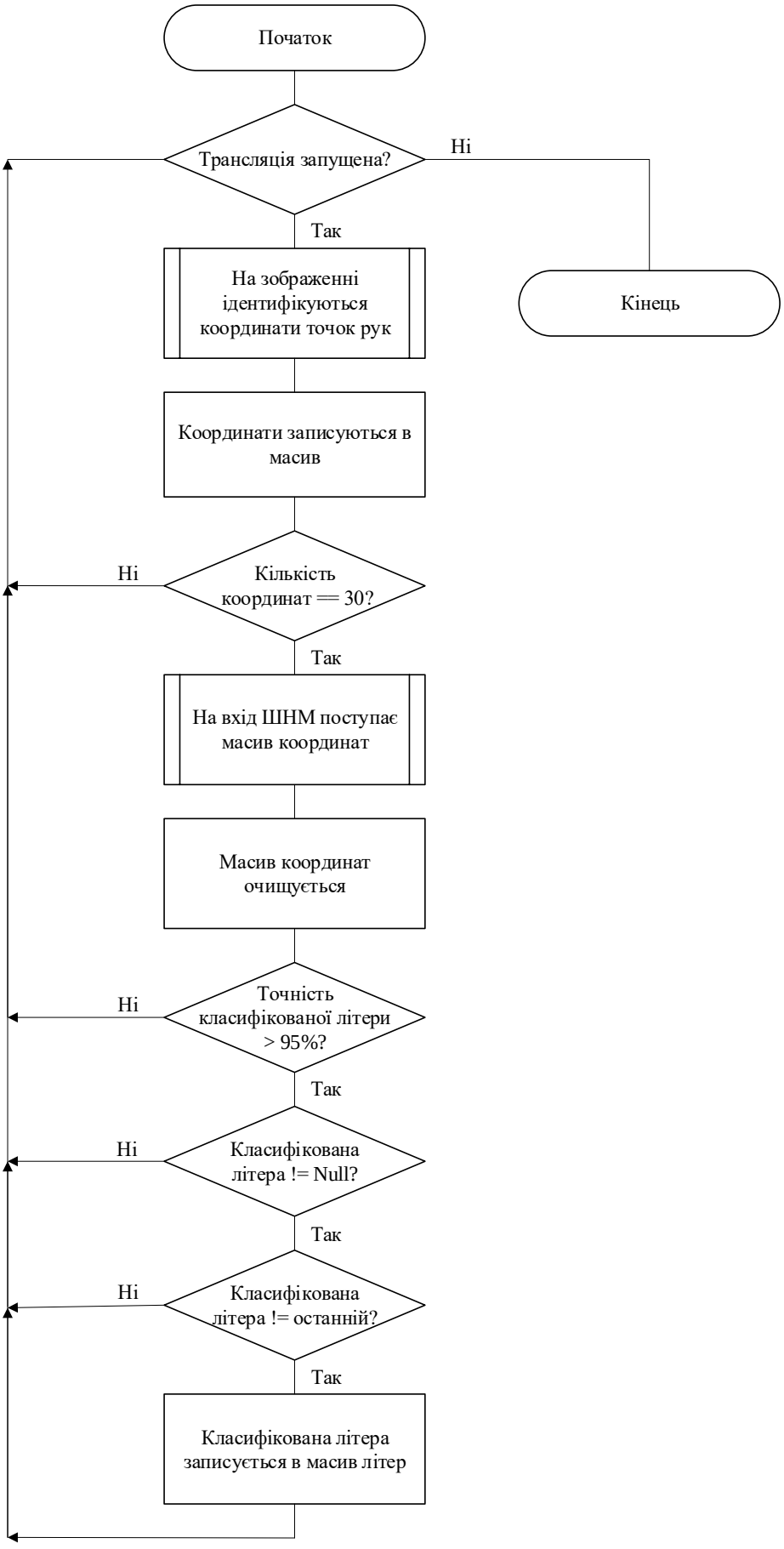


Рис. 2.9. Алгоритм відображення перекладу дактилеми

2.7. Інтерфейс користувача

Інтерфейс застосунку – веб-сторінка (рис. 2.10). Вона складається з блоку інформації про дактильний алфавіт, блоку трансляції з веб-камери та блоку перекладу.

Рис. 2.10. Інтерфейс застосунку

Блок трансляції (рис. 2.11) містить елемент параграф з текстом «Трансляція», передачу з веб-камери та кнопку з текстом «Почати/Зупинити». При переході на сторінку трансляція вимкнена, для її запуску потрібно натиснути на кнопку «Почати». Після цього текст кнопки змінюється на «Зупинити» і якщо натиснути на кнопку ще раз – трансляція зупиняється на останньому кадрі.

Блок перекладу (рис. 2.12) включає елемент параграф з текстом «Переклад», текстове поле виводу, та кнопку «Очистити». Після початку трансляції в текстове поле виводиться переклад класифікованого жесту. Якщо натиснути на кнопку «Очистити», поле очищується.

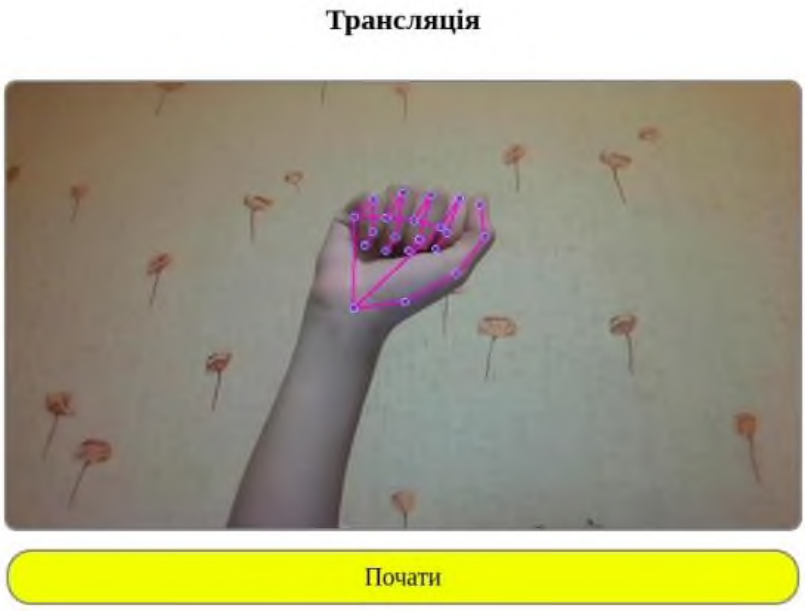


Рис. 2.11. Блок трансляції

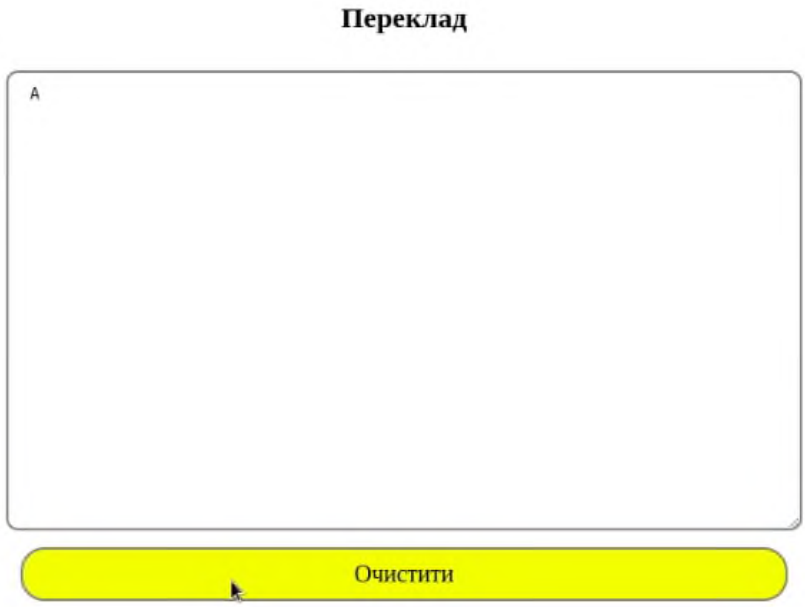


Рис. 2.12. Блок перекладу

2.8. Висновки до розділу

В даному розділі було розглянуто проєктування застосунку класифікації статичних та динамічних дактилем.

Розглянуто структуру додатку: модуль набору даних, модуль нейронної мережі та модуль класифікації дактилем.

Представлено подання даних, якими буде оперувати нейронна мережа: двовимірний масив, що складається з 30 кадрів кожен з яких містить 63 координати точок рук.

Описано архітектуру нейронної мережі, функції активації нейронів, оптимізатор та функція втрат:

– нейронна мережа складається з трьох шарів *LSTM* (довгої короткочасної пам'яті) та трьох повноз'язних шарів, що послідовно з'єднані;

– функції активації нейронів: перший–п'ятий шари – функція активації *ReLU*, шостий шар – функція активації *Softmax*;

– оптимізатор *Adam*;

– функція втрат – *Categorical Cross-Entropy Loss* (категоріальна перехресна ентропія).

Показано процес створення власного набору даних для навчання та тренування ШНМ.

Спроєктовано процес відображення перекладу класифікованої дактилеми: наведено блок-схему алгоритму та графічне відображення процесу.

Представлено інтерфейс користувача: веб-сторінка, що складається з блоків інформації про дактильний алфавіт, блоком трансляції з веб-камери, що включає кнопку керування потоком, блоком перекладу з можливістю очистити текстове поле для нового перекладу літер.

РОЗДІЛ 3

РОЗРОБКА ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

3.1. Ініціалізація проєкту

3.1.1. Інтерпретатор та віртуальне оточення

Основна частина коду проєкту написаний мовою програмування *Python*, версія 3.10. Її можна завантажити з головного сайту з розділу *Downloads* *https://www.python.org/downloads/* (рис. 3.1.).

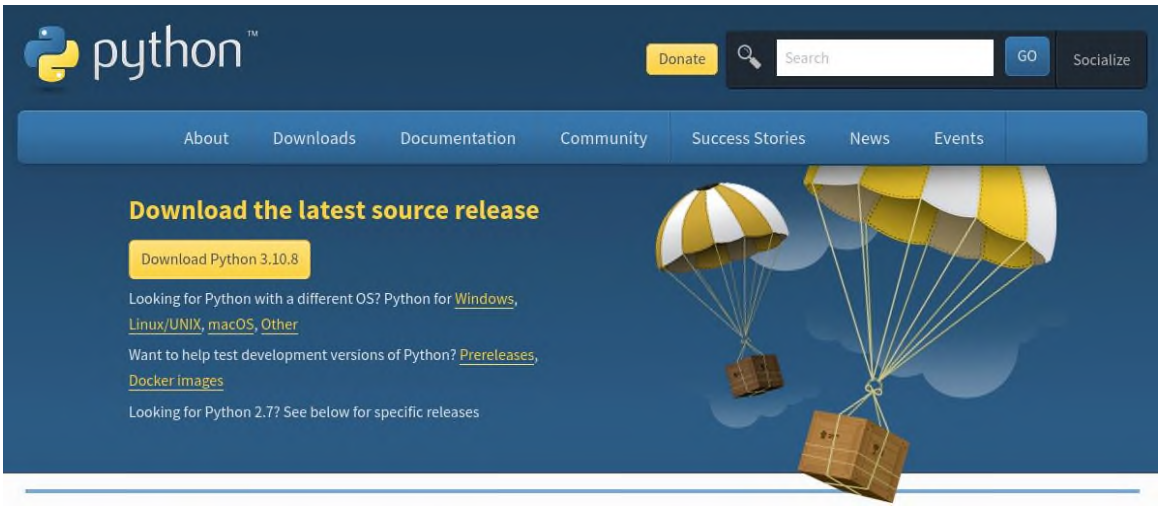


Рис. 3.1. Розділ завантаження *Python*

Разом із *Python* потрібно встановити систему керування пакетами *pip*. Застосунок розробляється на операційній системі *Ubuntu* 20.04, тому для установки *pip* для *Python 3* використовується наступна команда:

sudo apt install python3-pip.

В даному проєкті використовується менеджер залежностей *Pipenv*. З його допомогою можна створювати віртуальні середовища та керувати залежностями додатків. *Pipenv* вирішує низку проблем, які виникали при використанні *pip, virtualenv* та *requirements.txt.* Для своєї роботи менеджер використовує 2 файли: *Pipfile* (по суті заміна r*equirements.txt*) та *Pipfile.lock* (пов'язує версії пакетів, тим самим забезпечуючи додаткову безпеку). Завантаження менеджера залежностей *Pipenv* виконується командою:

pip install pipenv

Pipenv використовує свій власний формат файлу для опису залежностей проекту – *Pipfile*. Цей файл має формат *TOML*. У парі з *Pipfile* йде *Pipfile.lock*. Він має формат *JSON* та зберігає контрольні суми пакетів, які встановлюються у проєкт, що дає гарантію, що розгорнуті на різних машинах оточення будуть ідентичні один одному.

Сам проєкт було створено командою:

mkdir ukrainian-dactyl-recognition,

після цього було ініціалізовано віртуальне середовище у директорії проєкту:

pipenv --python 3.10.

У згенерований автоматично *Pipfile* були додані необхідні залежності, наразі файл виглядає наступним чином:

```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
tensorflow = "*"
tensorflow-gpu = "*"
opencv-python = "==4.5.5.64"
mediapipe = "*"
```

sklearn = ""*

matplotlib = ""*

numpy = ""*

flask = ""*

[dev-packages]

[requires]

python_version = "3.10".

У секції *[[source]]* перераховуються індекси пакетів. У секціях *[packages]* та *[dev-packages]* перераховуються залежності програми – ті, які потрібні для безпосередньої роботи програми (мінімум), і ті, які потрібні для розробки (запуск тестів, лінтери та інше). У секції *[requires]* вказана версія інтерпретатора, на якій цей застосунок може працювати.

Видалення та оновлення залежностей відбувається за допомогою команд *pipenv uninstall* та *pipenv update* відповідно.

3.1.2. Файли проєкту

Структура файлів та директорій проєкту *ukrainian-dactyl-recognition* має наступний вигляд (рис. 3.2):

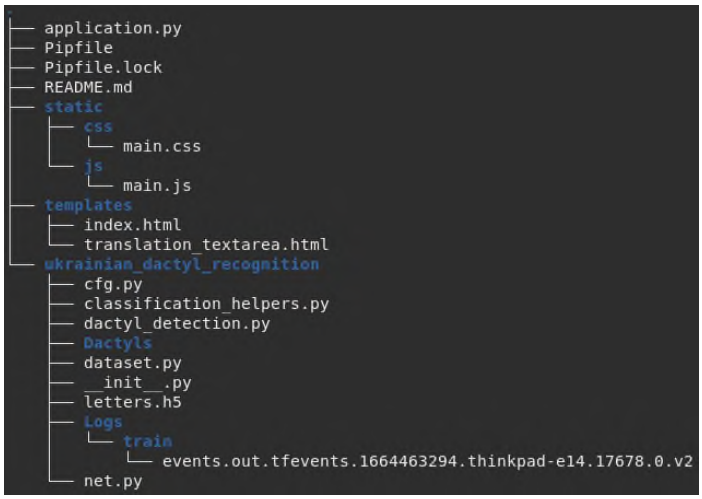


Рис. 3.2. Структура файлів та директорій проєкту

Опис файлів (у порядку як на рис. 3.2):

– Файл *main.py* – головний файл проєкту. Через нього відбувається запуск застосунку, а саме процесу класифікації дактилем.

– Файли *Pipfile* та *Pipfile.lock* містять дані про залежності проєкту.

– Файл *README.md* – інформація про проєкт у текстовому вигляді.

– Директорія *static* складається із статичних файлів:

– В папці *css* знаходиться головний файл зі стилями для веб-сторінки додатку;

– В папці *js* знаходиться головний файл із *jQuery AJAX* функціями для клієнт-серверної комунікації веб-сторінки.

– Директорія *templates* містить необхідні для роботи веб-сторінки базові шаблони:

– Файл *index.html* – розмітка веб-сторінки;

– Файл *translation_textarea.html* – розмітка текстового поля виводу перекладу жесту.

– Директорія *ukrainian_dactyl_recognition* – *Python* пакет проєкту, що складається з основних модулів застосунку:

– Файл *cfg.py* – файл з глобальними змінними застосунку;

– Файл *classification_helpers.py* – файл, що містить клас *ClassificationHelpers* з допоміжними функціями для визначення координат рук та їх відмальовки;

– Файл *dactyl_detection.py* – містить клас *DactylClassification*, що запускає процес класифікації дактилем;

– Файл *dactyls.h5* – збережена навчена нейронна мережа;

– Директорія *Dactyls* – створений датасет (на рис. 3.2. у скороченому вигляді, оскільки директорія містить значну кількість файлів та папок, її структура та опис були детально розглянуті у п. 2.5.2.);

– Файл *dataset.py* – містить клас *Dataset* з методами створення та підготовки датасету;

– Файл *__init__.py* – ініціалізація пакету *ukrainian_dactyl_recognition*;

– Директорія *Logs* – в ній зберігаються логи під час навчання нейронної мережі;

– Файл *net.py* – клас нейронної мережі *Net*.

Діаграма класів, які були згадані вище, їх структура та взаємодія наведена на рис. 3.3.

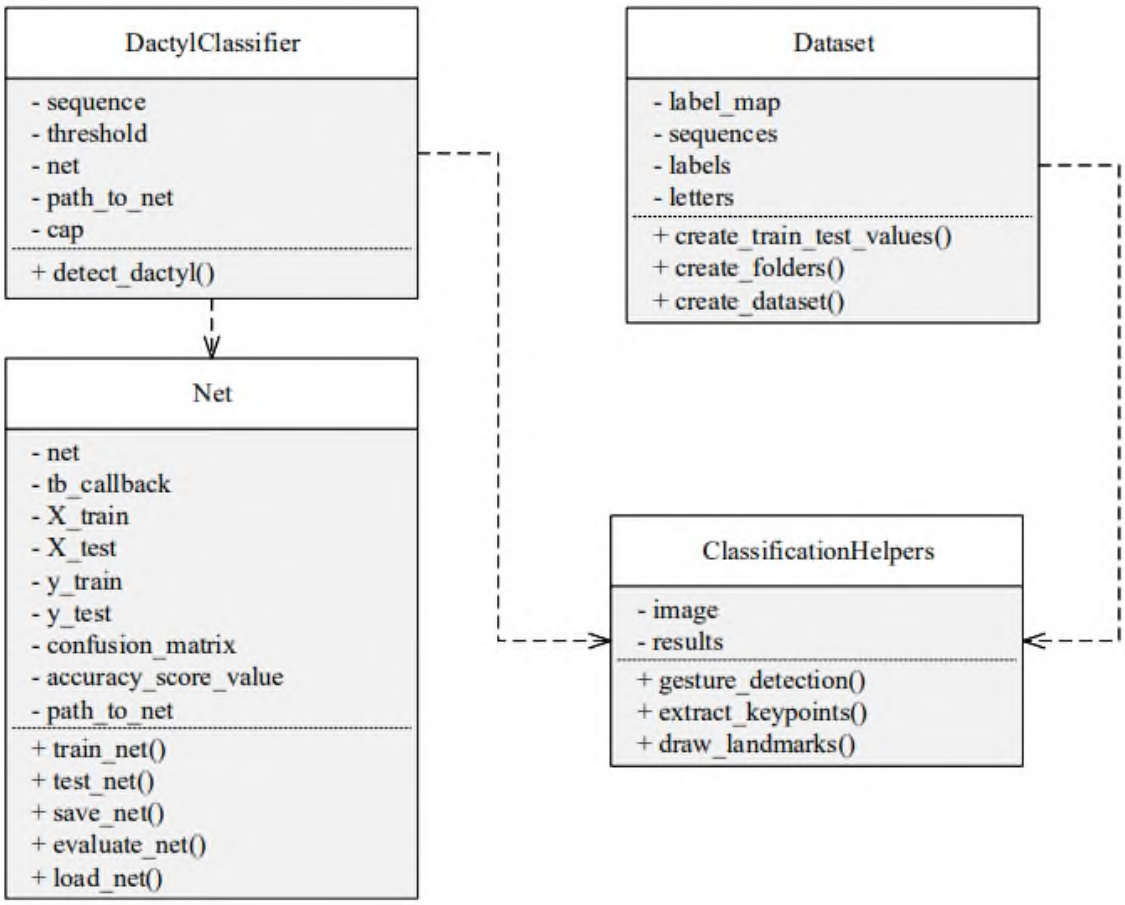


Рис. 3.3. Діаграма класів

3.2. Підготовка даних

Як зазначалося у пункті 2.2, бібліотека *Mediapipe* містить навчену ШНМ для розпізнавання рук. Із бібліотеки потрібно імпортувати модель *Holistic*, а також утиліти для відмальовки точок руки, бо надалі вони знадобляться для створення датасету та для класифікації дактилем:

mp_holistic = mp.solutions.holistic

mp_drawing = mp.solutions.drawing_utils.

Для виявлення рук на модель *Holistic* потрібно передати кадр с трансляції *OpenCV*. Проте, оскільки *OpenCV* використовує колірний канал у форматі *BGR*, потрібно конвертувати кадр з формату *BGR* в *RGB*, для збереження пам'яті кадр робиться неможливим для запису, далі він передається на модель, з неї отримується результат розпізнавання, а кадр стає знову можливим для запису та у форматі *BGR*. Наведені вище дії реалізовано у методі *gesture_detection* класу *ClassificationHelpers*, його вхідними параметрами є кадр з трансляції та модель *Holistic*, а вихідними – кадр та результат розпізнавання:

```
def gesture_detection(self):
    self.image = cv2.cvtColor(self.image, cv2.COLOR_BGR2RGB)
    self.image.flags.writeable = False
    self.results = self.model.process(self.image)
    self.image.flags.writeable = True
    self.image = cv2.cvtColor(self.image, cv2.COLOR_RGB2BGR)
    return self.image, self.results
```

Відмальовка точок рук реалізована у методі *draw_landmarks* класу *ClassificationHelpers*, що приймає на вхід кадр та результат розпізнавання:

```
def draw_landmarks(self):
    self.mp_drawing.draw_landmarks(
       self.image,
       self.results.left_hand_landmarks,
       self.mp_holistic.HAND_CONNECTIONS)
    self.mp_drawing.draw_landmarks(
       self.image,
       self.results.right_hand_landmarks,
       self.mp_holistic.HAND_CONNECTIONS).
```

Метод *gesture_detection* класу *ClassificationHelpers* повертає результат розпізнавання руки на кадрі, з якого можна виокремити координати точок кожної руки. Зберігатися вони будуть у одновимірному *numpy* масиві довжиною

63 (21 точка руки помножена на три виміри). Однак, у випадку, коли рука не була присутня у кадрі, при спробі вилучити її координати буде отримано виключення, для уникнення якого, масив координат у такому випадку буде заповнено шістдесятьма трьома нулями. Імплементовано у функції *extract_keypoints*, її вхідним параметром є результат розпізнавання руки на кадрі:

```
def extract_keypoints(results):
    rh = np.array([[res.x, res.y, res.z] for res in
    results.right_hand_landmarks.landmark]).flatten()
    if results.right_hand_landmarks else np.zeros(21 * 3)

    lh = np.array([[res.x, res.y, res.z] for res in
        results.left_hand_landmarks.landmark]).flatten()
    if results.left_hand_landmarks else np.zeros(21 * 3)
   return np.concatenate([rh, lh]).
```

Функції *gesture_detection*, *draw_landmarks* та *extract_keypoints* будуть визвані в середині циклу трансляції з веб-камери для кожного кадру.

3.3. Створення набору даних

Набір даних створюється в класі *Dataset*, в якому ініціалізуються наступні параметри:

директорія, в якій буде зберігатися датасет:

DATA_PATH = os.path.join('Dactyls'),

класи датасету:

letters = np.array([«А», «Б», «В», «Г», «Ґ», «Д», «Е», «Є», «Ж», «З», «И», «І», «Ї», «Й», «К», «Л», «М», «Н», «О», «П», «Р», «С», «Т», «У», «Ф», «Х», «Ц», «Ч», «Ш», «Щ», «Ь», «Ю», «Я», «*null*»])

кількість жестів для кожного класу:

no_sequences = 60

кількість кадрів для вираження одного жесту:

sequence_length = 30.

Далі потрібно створити папки для класів, жестів, кадрів датасету:

```
def create_folders(self):
    for letter in cfg.letters:
        for sequence in range(cfg.no_sequences):
            try:
                os.makedirs(os.path.join(cfg.DATA_PATH, letter, str(sequence)))
            except:
                pass
```

Забір набору даних відбувається у трьох вкладених циклах. Перший цикл проходиться по кожному класу датасету. Другий цикл виконує 60 ітерацій (кількість жестів в класі). Третій – 30 ітерацій (кількість кадрів). В середині циклів відбувається розпізнавання руки, відмальовка точок та вилучення координат точок. В кінці *numpy* масив з координатами зберігається у відповідну папку.

Для більш зручного збору даних, у вікні з трансляцією виведено назву класу, та номер поточного жесту. Також після забору 30 кадрів для одного жесту робиться пауза на 3 секунди, щоб встигнути змінити жест.

Збір датасету реалізовано в методі *create_dataset*:

```
def create_dataset(self):
        cap = cv2.VideoCapture(0)
        cap.set(3, 1280)
        cap.set(4, 720)
        with cfg.mp_holistic.Holistic(min_detection_confidence=0.5,
                                      min_tracking_confidence=0.5) as holistic:
            for letter in cfg.letters:
                for sequence in range(cfg.no_sequences):
                    for frame_num in range(cfg.sequence_length):
```

```
                ret, frame = cap.read()
                helper = ClassificationHelpers(frame, holistic)
                image, results = helper.gesture_detection()
                helper.draw_landmarks()
                keypoints = helper.extract_keypoints()

                cv2.putText(image,
                        f'Collecting frames video number {sequence}, {letter}',
                        (15, 12), cv2.FONT_HERSHEY_SIMPLEX,
                        0.5, (0, 0, 0), 1, cv2.LINE_AA)
                if frame_num == 0:
                   cv2.waitKey(2000)
                cv2.imshow('OpenCV Feed', image)
                np_path = os.path.join(cfg.DATA_PATH,
                                letter,
                                str(sequence),
                                str(frame_num))
                np.save(np_path, keypoints)

                if cv2.waitKey(10) & 0xFF == ord('q'):
                   break
            cap.release()
            cv2.destroyAllWindows()
```

Наступним кроком є попередня обробка датасету.

Для репрезентації кожного жесту створюється словник з мітками класів: *label_map = {label: num for num, label in enumerate(cfg.letters)}*.

Завантаження датасету відбувається наступним чином: в трьох вкладених циклах для одного жесту в один масив завантажуються масиви з координатами. В свою чергу, жест додається до масиву жестів, разом з ними в масив міток класу додається значення із словника *label_map*, ключем якого є клас поточного

жесту. Масив міток нормалізується за допомогою функції бібліотеки *Keras to_categorical().*

Після завантаження датасету, за допомогою функції *train_test_split*, його дані розбиваються на тренувальні та тестувальні. Відсоток розбиття обрано 20%.

Попередня обробка набору даних виконана у методі *create_train_test_values*, що повертає готовий до передачі на нейронну мережу датасет:

```
def create_train_test_values(self):
    for letter in cfg.letters:
        for sequence in range(cfg.no_sequences):
            window = []
            for frame_num in range(cfg.sequence_length):
                res = np.load(os.path.join(cfg.DATA_PATH,
                                           letter,
                                           str(sequence),
                                           f"{frame_num}.npy"))
                window.append(res)
            sequences.append(window)
            labels.append(label_map[letter])

    X = np.array(sequences)
    y = to_categorical(labels).astype(int)
    return train_test_split(X, y, test_size=0.2)
```

3.4. Розробка та навчання нейронної мережі

3.4.1. Розробка нейронної мережі

Для створення та навчання нейронної мережі застосовується бібліотека глибинного навчання *Keras*. Для початку, імпортуються модель *Sequential* та шари *LSTM* і *Dense*:

from keras.models import Sequential

from keras.layers import LSTM, Dense.

Нейронну мережу реалізовано у класі *Net*. В конструкторі класу створюється об’єкт моделі *Sequential*, ініціалізуються шари *LSTM* і *Dense*, створюються тренувальні та тестувальні дані викликом функції *create_train_test_values*, що описана вище. ШНМ складається з 6 шарів: з трьох *LSTM* та трьох *Dense*. Для кожного шару визначена кількість нейронів та функція активації нейрона, для першого шару вказана форма вхідних даних. Для першого та другого *LSTM* шару параметр *return_sequences* дорівнює *True*, оскільки при використанні шарів *LSTM*, що йдуть один за одним, на вхід наступного *LSTM* шару потрібно передавати повну послідовність прихованих станів кожного кроку часу із попереднього шару; для третього шару параметр *return_sequences* дорівнює *False*:

```
def __init__(self):
    self.model = Sequential()
    self.model.add(LSTM(64, return_sequences=True, activation='relu',
                        input_shape=(30, 126)))
    self.model.add(LSTM(128, return_sequences=True, activation='relu'))
    self.model.add(LSTM(64, return_sequences=False, activation='relu'))
    self.model.add(Dense(64, activation='relu'))
    self.model.add(Dense(32, activation='relu'))
    self.model.add(Dense(cfg.letters.shape[0], activation='softmax'))
    self.X_train, self.X_test, self.y_train, self.y_test =
create_train_test_values().
```

Метод *train_model* призначений для навчання нейронної мережі. Для цього використовується функція витрат *categorical_crossentropy*, оптимізатор *Adam*, а також опціональні метрики *categorical_accuracy*, що дозволяють відслідковувати точність ШНМ під час навчання. Вказуються тренувальні дані та їх мітки, кількість епох, що рівна 275, та зворотній зв'язок, завдяки якому можна контролювати показники нейронної мережі на будь якому етапі навчання:

```
def train_model(self):
    self.model.compile(optimizer='Adam',
                       loss='categorical_crossentropy',
                       metrics=['categorical_accuracy'])
    self.model.fit(self.X_train,
                   self.y_train,
                   epochs=300,
                   callbacks=[tb_callback])
```

Метод *test_model* тестує навчену модель на виокремлених для тестування даних датасету. Функція тренування повертає масив передбачених дактилем.

```
def test_model(self):
    res = self.model.predict(self.X_test)
```

Метод save_model зберігає ваги навченої нейронної мережі у вказаний файл з розширенням .h5.

```
def save_model(self):
    self.model.save('letters.h5').
```

Із бібліотеки *Sklearn* імпортуються функції *multilabel_confusion_matrix* та *accuracy_score*. Функція *multilabel_confusion_matrix* обчислює матрицю плутанини з кількома мітками для класів, щоб оцінити точність класифікації, і повертає матриці плутанини для кожного класу або зразка. Функція *accuracy_score* оцінює точність класифікації:

```
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score
```

Матриця плутанини – це таблиця з 4 різними комбінаціями прогнозованих і фактичних значень (рис. 3.4) – *True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN).*

Рис. 3.4. Матриця плутанини

Вірний позитивний результат (*TP*): значення позитивне, і це правда.

Вірний негативний результат (*TN*): значення негативне, і це правда.

Помилково позитивний результат (*FP*): значення позитивне, але воно невірне.

Помилково негативний результат (*FN*): значення негативне, і це невірно.

Метод *evaluate_model* оцінює продуктивність навченної нейронної мережі. Спочатку отримуються передбачувані класи тестувальних даних датасету. Передбачення повертаються у вигляді бінарної матриці, тому вони конвертуються у одновимірний масив, де передбачена мітка класу є числом, що представляє клас. До такого ж виду конвертуються і мітки тестувальних даних. У функцію *multilabel_confusion_matrix* передаються мітки передбачуваних класів та тестувальних даних.

```
def evaluate_model(self):
    yhat = self.model.predict(self.X_test)
    ytrue = np.argmax(self.y_test, axis=1).tolist()
    yhat = np.argmax(yhat, axis=1).tolist()
    confusion_matrix = multilabel_confusion_matrix(ytrue, yhat)
    accuracy_score_value = accuracy_score(ytrue, yhat).
```

3.4.2. Навчання нейронної мережі

Після початку навчання нейронної мережі, у консоль виводиться інформація про кожну епоху навчання, така як час епохи, втрата та точність (рис. 3.5.):

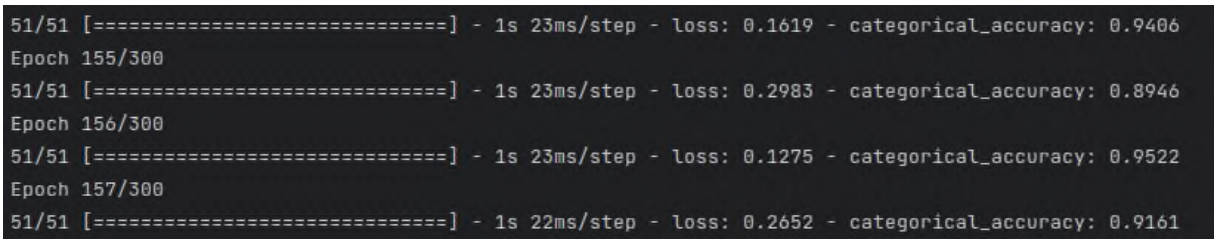


Рис. 3.5. Логи навчання нейронної мережі

Процес навчання можна відслідковувати за допомогою *TensorBoard*. Для цього потрібно перейти до папки *Logs/train* і виконати команду:

tensorboard --logdir='Logs/train',

після чого у консолі з'явиться посилання, за яким потрібно перейти (рис. 3.6-3.7):

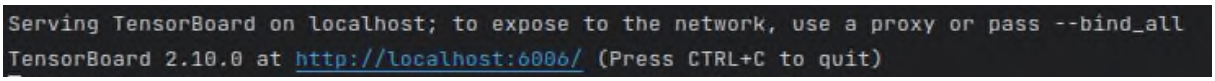


Рис. 3.6. Посилання на *TensorBoard*

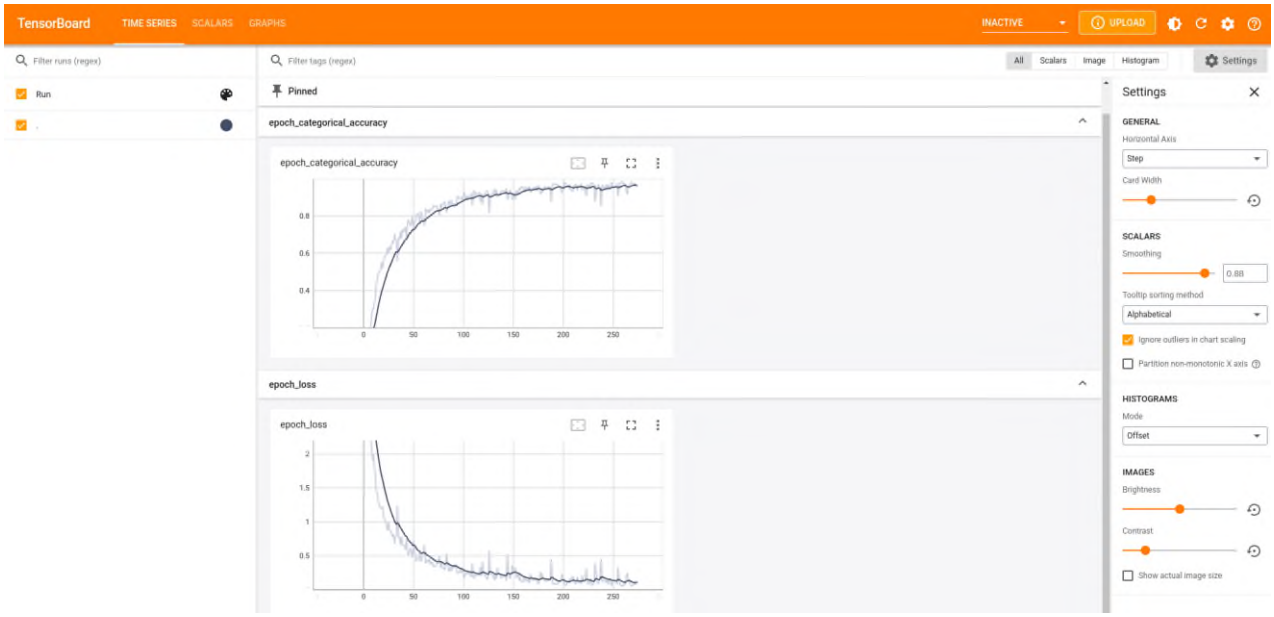


Рис. 3.7. Головна сторінка *TensorBoard*

На головній сторінці *TensorBoard* можна побачити два графіки: графік залежності точності від епохи (рис. 3.8.) та графік залежності втрат від епохи (рис. 3.9).

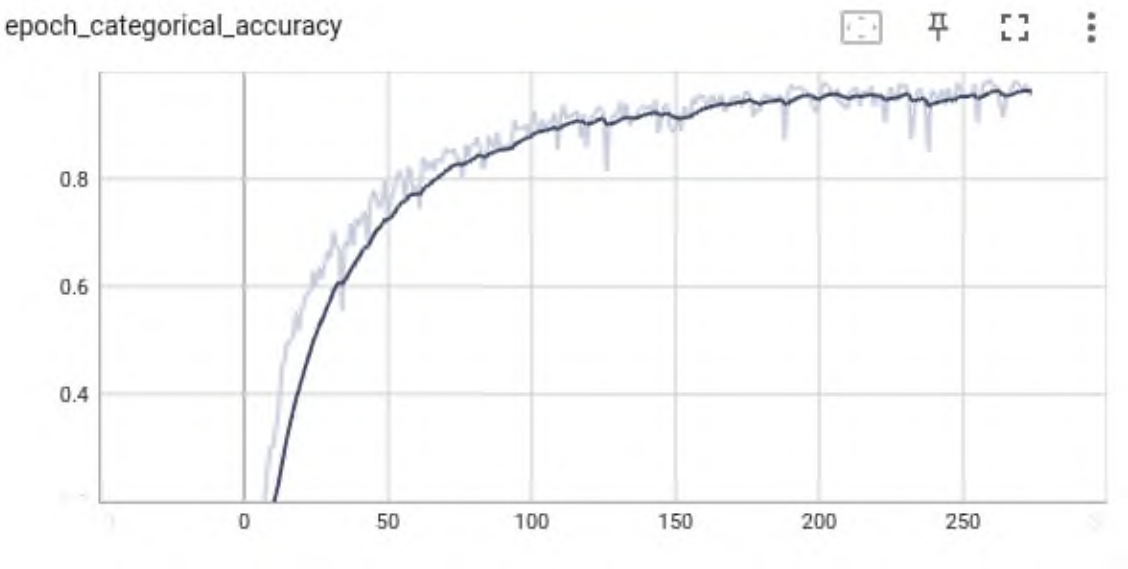


Рис. 3.8. Графік залежності точності від епохи (275 епох)

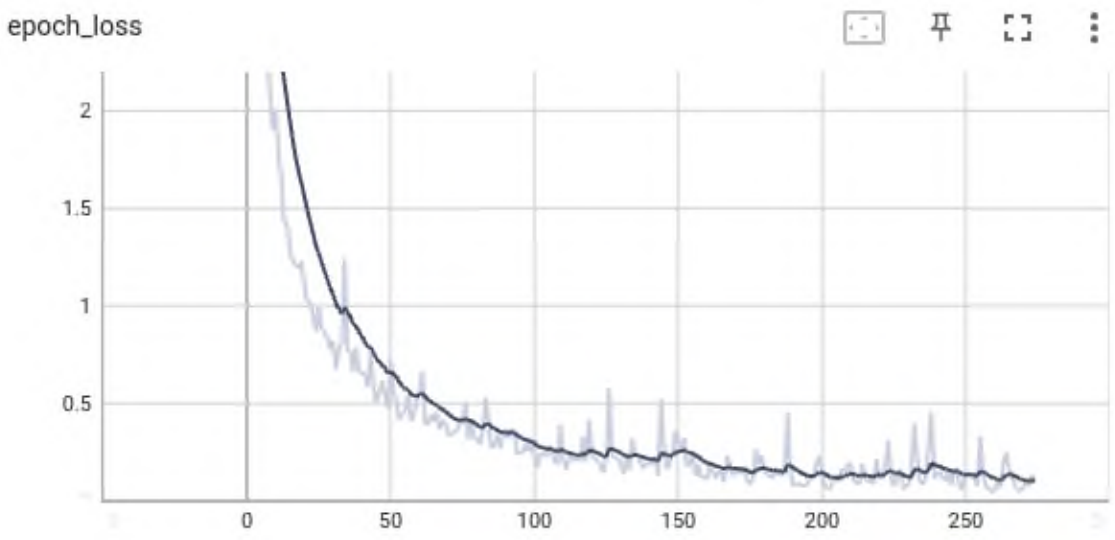


Рис. 3.9. Графік залежності втрат від епохи (275 епох)

Також на *TensorBoard* можна побачити граф нейронної мережі (рис. 3.10.).

В кінці навчання в консоль виводиться підсумкова таблиця параметрів ШНМ (рис. 3.11).

Таблиця складається з трьох стовпців: типу шару, форми вихідних даних та кількості параметрів. Перший шар *LSTM*, як було вказано вище, має форму вхідних даних (30, 126), форму вихідних даних – (*None*, 30, 64), кількість параметрів – 48896. Вихідні дані мають форму тривимірного масиву, де перший вимір вказує на кількість зразків у партії, наданій шару *LSTM*; другий вимір – це кількість часових кроків у вхідній послідовності; третій вимір – це розмірність вихідного простору, визначеного параметром нейронів у реалізації *Keras LSTM*.

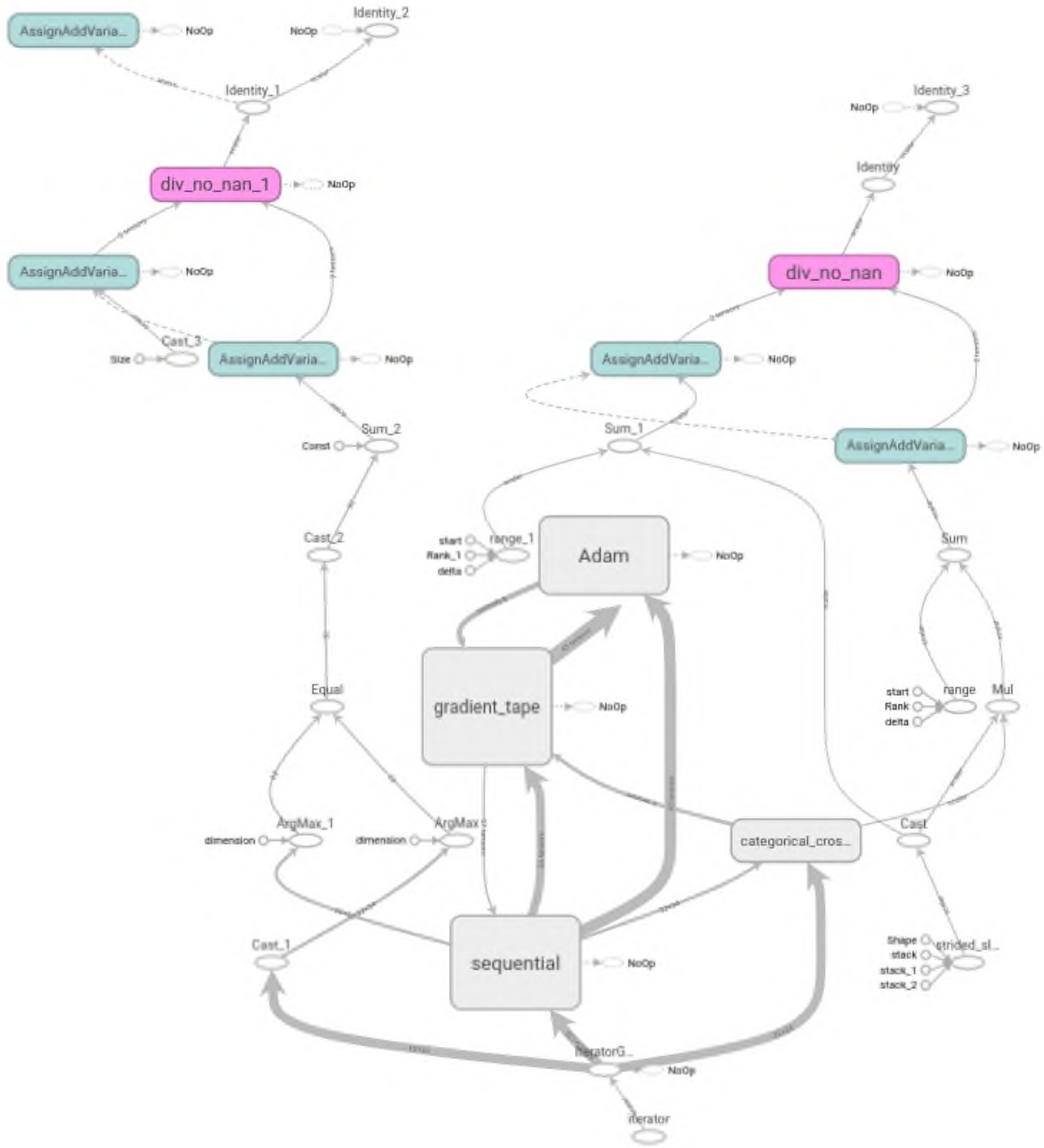
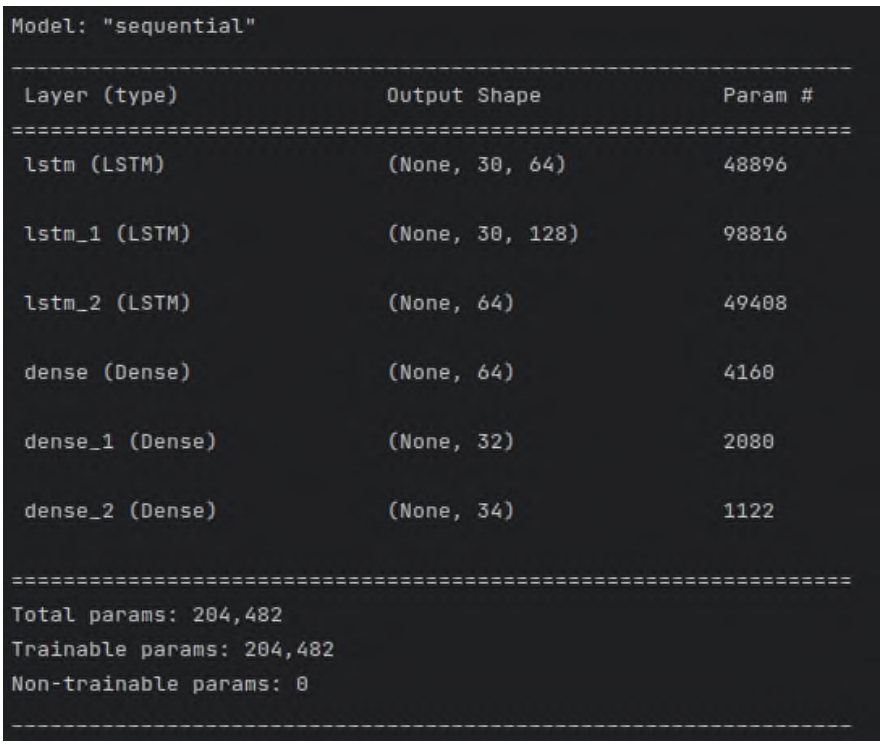


Рис. 3.10. Граф нейронної мережі, побудований у *TensorBoard*



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 64)            48896

 lstm_1 (LSTM)               (None, 30, 128)           98816

 lstm_2 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 34)                1122

=================================================================
Total params: 204,482
Trainable params: 204,482
Non-trainable params: 0
_________________________________________________________________
```

Рис. 3.11. Підсумкова таблиця параметрів нейронної мережі

Кількість параметрів *LSTM* шару визначається за формулою:

$num\_params_{LSTM} = 4 \times ((size\_of\_input + 1) \times size\_of\_output + size\_of\_output^2)$,

де *size_of_input* – розмірність вхідного простору;

size_of_output – розмірність вихідного простору;

+1 – зсув (*bias*).

Кількість параметрів для кожного *LSTM* шару визначається наступним чином:

$num\_params_{LSTM1} = 4 \times ((126 + 1) \times 64 + 64^2) = 48896$

$num\_params_{LSTM2} = 4 \times ((64 + 1) \times 128 + 128^2) = 98816$

$num\_params_{LSTM3} = 4 \times ((128 + 1) \times 64 + 64^2) = 49408$.

Кількість параметрів повнозв’язних шарів можна визначити за формулою:

$num\_params_{Dense} = size\_of\_input \times num\_of\_neurons + bias\_value$,

де *size_of_input* – розмірність вхідного простору;

num_of_neurons – кількість нейронів в шарі;

bias_value – значення зсувів в шарі.

Кількість параметрів для кожного *Dense* шару наступна:

$num\_params_{Dense1} = 64 \times 64 + 64 = 4160$

$num\_params_{Dense2} = 64 \times 32 + 32 = 2080$

$num\_params_{Dense3} = 32 \times 34 + 34 = 1122.$

Загальна кількість параметрів дорівнює 204482.

Також, після завершення навчання нейронної мережі, можна побачити точність (рис. 3.12), матрицю плутанини (рис. 3.13) та результати тестування (рис. 3.14).

```
Точність: 0.8848039215686274
```

Рис. 3.12. Точність навченної нейронної мережі

```
========================
Результат: А
Очікуваний результат: Ь
========================
Результат: Ґ
Очікуваний результат: Ґ
========================
Результат: И
Очікуваний результат: И
========================
Результат: Б
Очікуваний результат: Б
========================
Результат: С
Очікуваний результат: С
========================
Результат: Є
Очікуваний результат: Є
========================
Результат: Я
Очікуваний результат: Я
========================
Результат: Й
Очікуваний результат: Й
========================
Результат: Ї
Очікуваний результат: Ю
========================
Результат: Д
Очікуваний результат: Д
========================
```

Рис. 3.13. Результати тестування (перші 10)

```
[395   2]
[  0  11]]

[393   1]
[  3  11]]

[397   1]
[  1   9]]

[394   3]
[  3   8]]

[390   3]
[  2  13]]

[393   2]
[  0  13]]

[395   0]
[  0  13]]

[397   0]
[  0  11]]

[400   1]
[  0   7]]

[396   1]
[  1  10]]
```

Рис. 3.14. Матриці плутанини (перші 10)

3.5. Реалізація класифікації дактилем

Класифікація дактлем реалізована у класі *DactylClassification* у методі *detect_dactyl* і починається з ініціалізації змінних: списку координат, порогу відсотку вірогідності передбачуваного класу (90%), останньої класифікованої літери (початкове значення «*null*») та ШНМ, ваги якої завантажуються із файлу.

Класифікація відбувається в нескінченному циклі *while*, параметр *is_feed_on* відповідає за початок та припинення трансляції: якщо *is_feed_on* дорівнює *True* – трансляція активна, в іншому випадку трансляція зупинена. Як і при створені датасету, на кадрах розпізнаються руки, вилучаються координати їх точок, які згодом відмальовуються на кадрі. Координати точок рук додаються до списку координат, якщо довжина списку рівна 30, список передається на нейронну мережу, відбувається класифікація жесту, список координат очищується, якщо вірогідність прогнозованої дактилеми більше порогового значення, тоді перевіряється, чи не є прогнозованим класом клас «*null*» та чи не дорівнює він попередньому класу. У такому випадку дактилема додається до списку дактилем, останньою літерою стає поточна.

На кожній ітерації циклу кадр, за допомогою функції *imencode*(), кодується у потокові дані та зберігається у кеш-пам’яті. Це потрібно для стиснення форматів даних зображень, щоб полегшити передачу по мережі. Далі закодований кадр оператором *yield* передається на веб-сторінку (клієнт). Таким чином реалізовано передачу трансляції з веб-камери на веб-сторінку.

```
def detect_dactyl(self):
    cap = cv2.VideoCapture(0)
    cap.set(3, 1280)
    cap.set(4, 720)
    with cfg.mp_holistic.Holistic(min_detection_confidence=0.5,
                                  min_tracking_confidence=0.5) as holistic:

    while cap.isOpened():
```

```
if cfg.is_feed_on:
    ret, frame = cap.read()
    helper = ClassificationHelpers(frame, holistic)
    image, results = helper.gesture_detection()
    helper.draw_landmarks()
    keypoints = helper.extract_keypoints()
    self.sequence.append(keypoints)

    if len(self.sequence) == 30:
        res = self.model.predict(np.expand_dims(self.sequence, axis=0))[0]
        self.sequence = []
        predicted_dactyl = cfg.letters[np.argmax(res)]

        if res[np.argmax(res)] > self.threshold:
            if predicted_dactyl != self.last_letter:
                if predicted_dactyl != 'null':
                    cfg.sentence.append(predicted_dactyl)
                self.last_letter = predicted_dactyl

    ret, buffer = cv2.imencode('.jpg', image)
    image = buffer.tobytes()

    yield b'--frame\r\nContent-Type: image/jpeg\r\n\r\n' + image + b'\r\n'
```

3.6. Розробка серверної частини застосунку

Серверна частина застосунку реалізована засобами фреймворку *Flask*. Вона складається з таких ендпоінтів:

– «/» рендерить головний *html* шаблон.

```
app = Flask(__name__)
```

```
@app.route('/')
def index():
    return render_template('index.html')
```

– «/*stream*», метод *GET*, покадрово передає трансляцію з веб-камери на веб-сторінку за допомогою заголовка *Content-Type multipart/x-mixed-replace*.

```
@app.route('/stream')
def stream():
    return Response(detect_dactyl(),
                    mimetype='multipart/x-mixed-replace; boundary=frame')
```

– «/*translation*», метод *GET*, рендерить шаблон *translation_textarea.html*, і передає в нього рядок з перекладом розпізнанних дактилем.

```
@app.route('/translation')
def translation():
    return jsonify('', render_template('translation_textarea.html',
                                       translation=''.join(cfg.sentence)))
```

– «/*clear_textarea*», метод *DELETE*, очищає текстове поле з перекладом та рендерить шаблон *translation_textarea.html* з пустим перекладом.

```
@app.route('/clear_textarea', methods=['DELETE'])
def clear_textarea():
   cfg.sentence = []
   return jsonify('', render_template('translation_textarea.html', translation=''))
```

– «/*erase_letter*», метод *DELETE*, видаляє із текстового поля з перекладом останню літеру та рендерить шаблон *translation_textarea.html* з оновленим перекладом.

```
@application.route('/erase_letter', methods=['DELETE'])
def erase_letter():
    if len(cfg.sentence) > 0:
```

```
        cfg.sentence.pop()
    return jsonify('', render_template('translation_textarea.html',
                                       translation=''.join(cfg.sentence)))
```

– «/*start_pause_webcam*», метод *PUT*, починає або зупиняє трансляцію та повертає відповідну назву для кнопки, приймає джейсон з ключем *btn_text*, за яким визначає, яку назву потрібно повернути. Повертає джейсон з ключем *btn_text* та новим текстом кнопки.

```
@app.route('/start_pause_webcam', methods=[PUT])
def start_pause_webcam():
    cfg.is_feed_on = not cfg.is_feed_on
    btn_text = request.get_json()['btn_text']
    btn_text = "Зупинити" if btn_text == "Почати" else "Почати"
    return jsonify({'btn_text': btn_text}).
```

3.7. Розробка веб-сторінки застосунку

3.7.1. Розмітка веб-сторінки

Веб-сторінка застосунку складається з трьох основних блоків, що реалізовані у файлі *index.html*: визначення дактильного алфавіту, трансляція з веб-камери та переклад.

Блок з визначенням містить заголовок першого рівня з текстом «Класифікація українського дактильного алфавіту», а також параграф із самим визначенням. Блок належить до класу *intro*.

```
<div class="intro">
    <h1>Класифікація українського дактильного алфавіту</h1>
    <p>Український дактильний алфавіт – допоміжна система…</p>
</div>.
```

Контейнер класу *grid-container* включає блоки трансляції з веб-камери та перекладу.

Блок трансляції належить до класу *web-cam*, та складається з заголовку другого рівня «Трансляція», вбудованого зображення, що відображає трансляцію з веб-камери, та кнопки з текстом «Почати/Зупинити» з класом *button*, ідентифікатором *button_start*.

```
<div class="web-cam">
    <h2>Трансляція</h2>
    <img src="{{ url_for('video') }}"/>
    <button class="button" id="button_start">Почати</button>
</div>
```

Блок перекладу має в собі заголовок другого рівня «Переклад», кнопку з текстом «Очистити», кнопку з текстом «Видалити літеру» та контейнер з ідентифікатором *translation_textarea*, що містить текстове поле, куди виводиться переклад. Цей контейнер також винесено окремо до файлу *translation_textarea.html* для того, щоб замінювати ним контейнер у файлі *index.html* при виводі перекладу.

```
<div class="translation">
    <h2>Переклад</h2>
    <div id="translation_textarea">
       <textarea readonly class="textarea" > {{translation}} </textarea>
    </div>
    <button class="button2" id="button_clear" >Очистити</button>
    <button class="button2" id="button_erase">Видалити літеру</button>
</div>.
```

Розмічена веб-сторінка виглядає так (рис. 3.15):

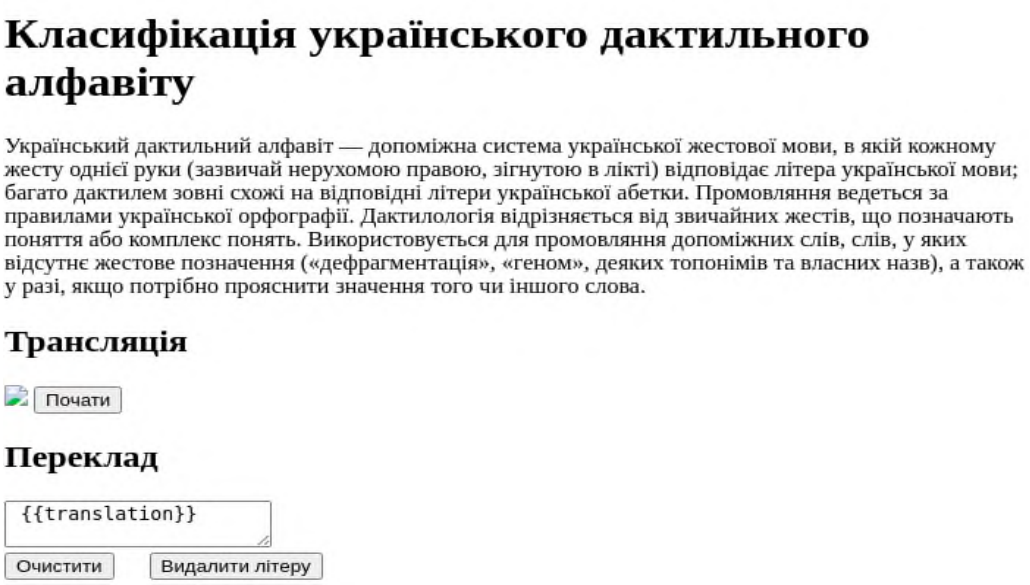


Рис. 3.15. Розмічена веб-сторінка застосунку

3.7.2. Стилі веб-сторінки

Стилі веб-сторінок перераховані у файлі *main.css*. Клас *intro* має такі властивості: колір фону – жовтий; зовнішній відступ: зверху 10 пікселів, справа та зліва 70 пікселів, знизу 50 пікселів; внутрішній відступ – 30 пікселів, рамка: ширина – 2 пікселі, колір – сірий, радіус – 10 пікселів.

```
.intro {
  background-color: yellow;
  margin: 10px 70px 50px 70px;
  padding: 30px;
  border-radius: 10px;
  border: 2px solid gray;
}
```

Для елементу <*body*> визначено сімейство шрифту:

```
body {
  font-family: "Times New Roman", Times, serif;
}.
```

Текст параграфу має сірий колір та розмір 20 пікселів:

```
p {
  color: gray;
  font-size: 20px;
}.
```

Ширина вбудованого зображення 640 пікселів, висота – 360 пікселів, рамка: ширина – 2 пікселі, колір – сірий, радіус – 10 пікселів, зовнішній відступ – 15 пікселів:

```
img {
  width: 640px;
  height: 360px;
  border: 2px solid gray;
  border-radius: 10px;
  margin: 15px;
}.
```

Клас *textarea* має властивості: ширина – 640 пікселів, висота – 360 пікселів, рамка: ширина – 2 пікселі, колір – сірий, радіус – 10 пікселів, зовнішній відступ – 10 пікселів, внутрішній відступ – 10 пікселів:

```
.textarea {
  width: 640px;
  height: 360px;
  border: 2px solid gray;
  border-radius: 10px;
  margin: 10px;
  padding: 10px;
}.
```

Клас *grid-container* визначає звичайний контейнер-сітку з двома колонками автоматичної ширини:

```
.grid-container {
   display: grid;
   grid-template-columns: auto auto;
```

}.

Для класів *web-cam* та *translation* характерне центральне вирівнювання відносно колонки:

```
.web-cam {
   display: flex;
   align-items: center;
   flex-direction: column;
}

.translation {
   display: flex;
   align-items: center;
   flex-direction: column;
}.
```

Для класу кнопки «Почати» властиве наступне: колір – жовтий; рамка: ширина – 2 пікселі, колір – сірий, радіус – 10; колір тексту – чорний; сімейство шрифту – «*Times New Roman*», *Times, serif*; розмір шрифту – 20 пікселів:

```
.button {
  background-color: yellow;
  border: 2px solid gray;
  color: black;
  width: 640px;
  height: 45px;
  font-family: "Times New Roman", Times, serif;
  text-align: center;
  font-size: 20px;
  border-radius: 10px;
}.
```

Клас кнопки *button2* має майже ті самі властивості, що і клас *button* за виключенням ширини кнопки – 310 пікселів.

```
.button2 {
  background-color: yellow;
  border: 2px solid gray;
  color: black;
  width: 310px;
  height: 45px;
  font-family: "Times New Roman", Times, serif;
  text-align: center;
  text-decoration: none;
  font-size: 20px;
  border-radius: 20px;
}
```

Зі стилями веб-сторінка має вигляд (рис. 3.16):

Рис. 3.16. Веб-сторінка застосунку із стилями

3.7.3. Асинхронні *HTTP* запити

Асинхронні запити *jQuery AJAX* імплементовані у файлі *main.js*. *AJAX* використовується для обміну даних із сервером без оновлення сторінки.

Для оновлення тестового поля, для відображення нового перекладу, кожну секунду виконується *GET* запит на ендпоінт «*/translation*». Повернений відмальований блок класу *translation_textarea* із файлу *translation_textarea.html* замінює відповідний йому блок у файлі *index.html*.

```
$(function(){
   window.setInterval(function(){
      updateTranslation()
   }, 1000)

  function updateTranslation(){
    $.ajax({
      url: "/translation",
      type: "GET",
      dataType: "json",
      success: function(data){
         $(translation_textarea).replaceWith(data)
      }
    });
  }
});
```

Клік по кнопці «Очистити» ініціює *DELETE* запит на ендпоінт *«/clear_textarea».*

```
$(document).on('click','#button_clear',function(){
    $.ajax({
      url: "/clear_textarea",
      type: "DELETE"
```

```
    });
});
```

При натисканні на кнопку «Видалити літеру» на ендпоінт «/*erase_letter*» відправляється *DELETE* запит, що видаляє останню класифіковану літеру.

```
$(document).on('click','#button_erase',function(){
    $.ajax({
      url: "/erase_letter",
      type: "DELETE"
    });
});
```

Клік по кнопці «Почати/Зупинити» відправляє джейсон з ключем «*btn_text*» та значенням, що є текстом даної кнопки на «/*start_pause_webcam*», метод *PUT*. В разі успішної відповіді текст кнопки оновлюється тим, що прийшов назад у джейсоні.

```
$(document).on('click','#button_start',function(){
    $.ajax({
      url: "/start_pause_webcam",
      type: "PUT",
      contentType: "application/json",
      dataType: 'json',
      data: JSON.stringify({
         btn_text: $(this).text()
      }),
      success: function(response){
         $("#button_start").text(response.btn_text)
      }
    });
});
```

3.8. Система контролю версій

3.8.1. Ініціалізація локального репозиторію

В проєкті використовується система контролю версій *git*, репозиторій розміщено на *GitHub* (*https://github.com/IrinaBalitskaya/ukrainian-dactyl-recognition*).

Щоб ініціалізувати *git* в проєкті (рис. 3.17) потрібно перейти до відповідної директорії та виконати команду, що створить пустий локальний репозиторій:

git init

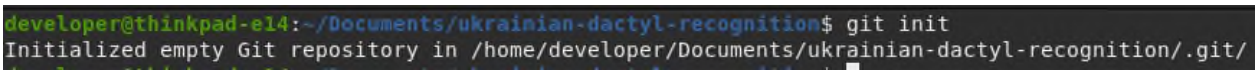


Рис. 3.17. Ініціалізація *git*

Команда *git status* виводить список файлів, що не відстежуються або були змінені, видалені, додані (рис. 3.18):

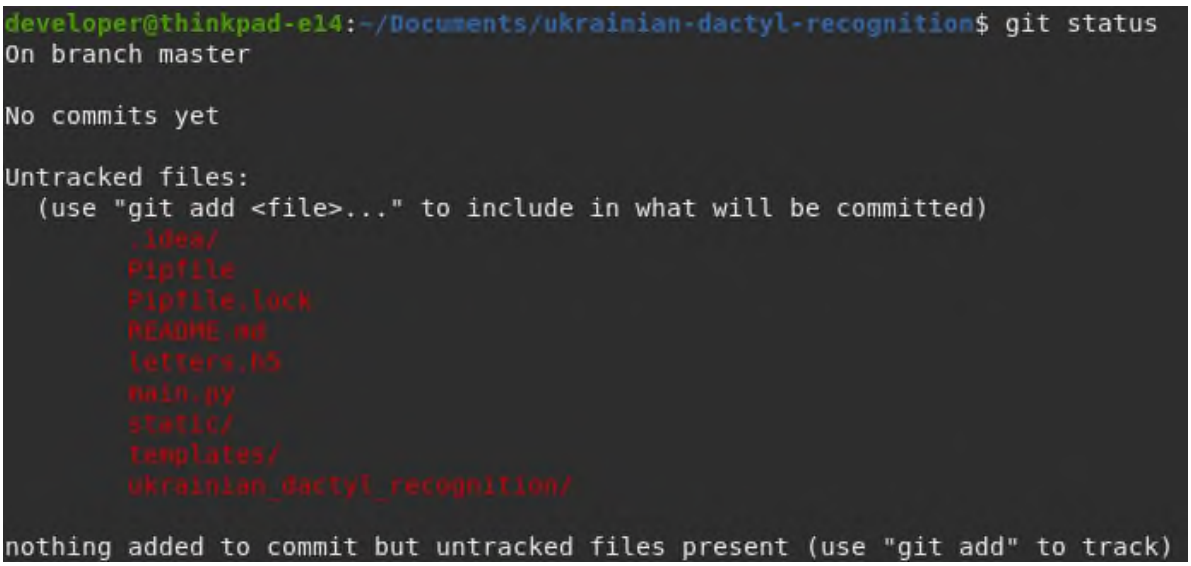


Рис. 3.18. Вивід команди *git status*

Додавання файла до проміжного середовища виконується командою *git add* ім’я_файлу, після цього файл готовий до коміту, обов’язково з повідомленням: *git commit -m* «повідомлення» (рис. 3.19). Коміти допомагають орієнтуватися у змінах в проєкті.

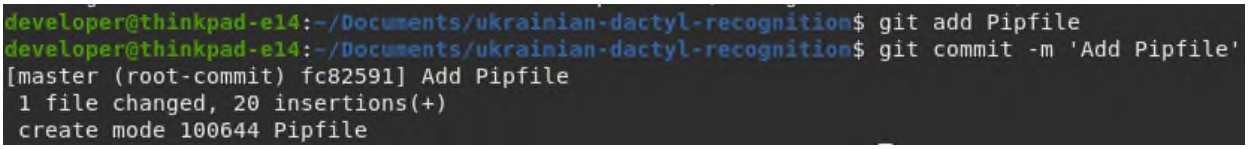


Рис. 3.19. Додавання файла до проміжного середовища

Історія комітів переглядається командою *git log* (рис. 3.20):

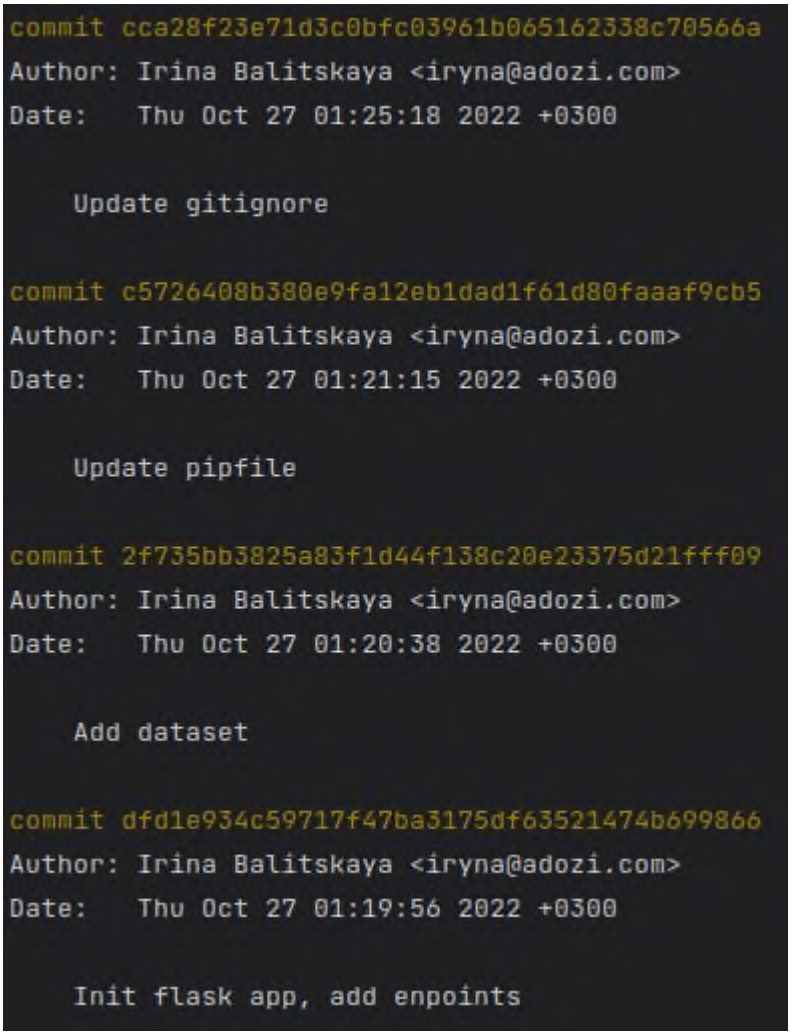


Рис. 3.20. Історія комітів

3.8.2. Створення віддаленого репозиторію

Для початку потрібно перейти на сайт *github.com*, увійти в обліковий запис або зареєструватися, та натиснути «Новий репозиторій» (рис. 3.21):

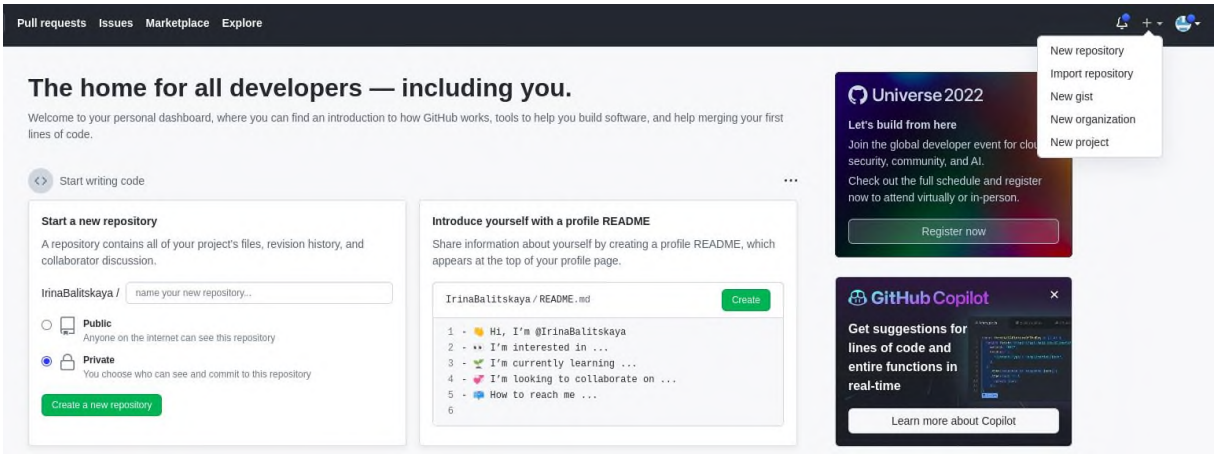


Рис. 3.21. Головна сторінка *GitHub*

На сторінці створення потрібно ввести ім’я репозиторію (повинно бути унікальним) та заповнити інші дані, після чого натиснути «Створити репозиторій» (рис. 3.22):

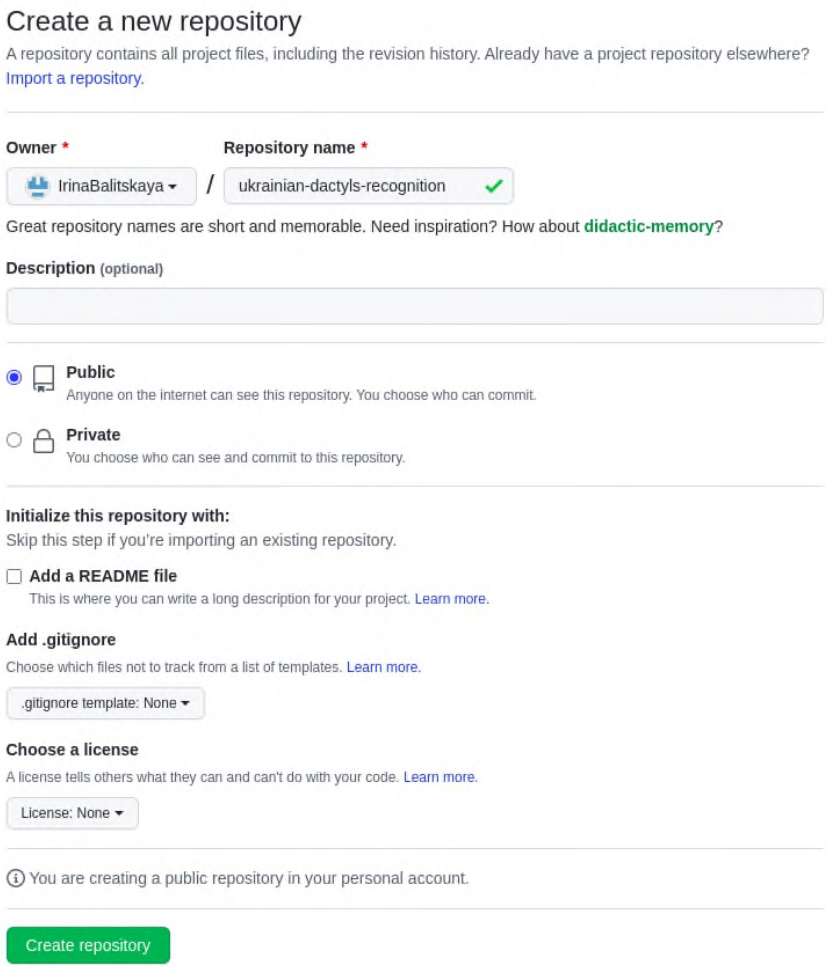


Рис. 3.22. Створення репозиторію на *GitHub*

Для зв'язки віддаленого репозиторію з локальним виконується клонування командою:

git clone https://github.com/IrinaBalitskaya/ukrainian-dactyl-recognition.git.

Після клонування коміти локального репозиторія надсилаються у віддалений командою *git push* (рис. 3.23).

```
(diploma) developer@thinkpad-e14:~/Documents/diploma/ukrainian-dactyl-recognition$ git push
Enumerating objects: 58971, done.
Counting objects: 100% (58971/58971), done.
Delta compression using up to 8 threads
Compressing objects: 100% (58945/58945), done.
Writing objects: 100% (58966/58966), 24.67 MiB | 1.03 MiB/s, done.
Total 58966 (delta 56438), reused 46 (delta 15)
remote: Resolving deltas: 100% (56438/56438), completed with 3 local objects.
To https://github.com/IrinaBalitskaya/ukrainian-dactyl-recognition.git
   b12722d..1dff31f  master -> master
```

Рис. 3.23. Пуш локальних комітів у віддалений репозиторій

3.9. Розгортання застосунку

Для розгортання застосунку використано сервіс *AWS Elastic Beanstalk*. Потрібно перейти на сайт *https://aws.amazon.com/* та авторизуватися. На головній консолі (рис. 3.24), у пошуку потрібно ввести *Elastic Beanstalk*.

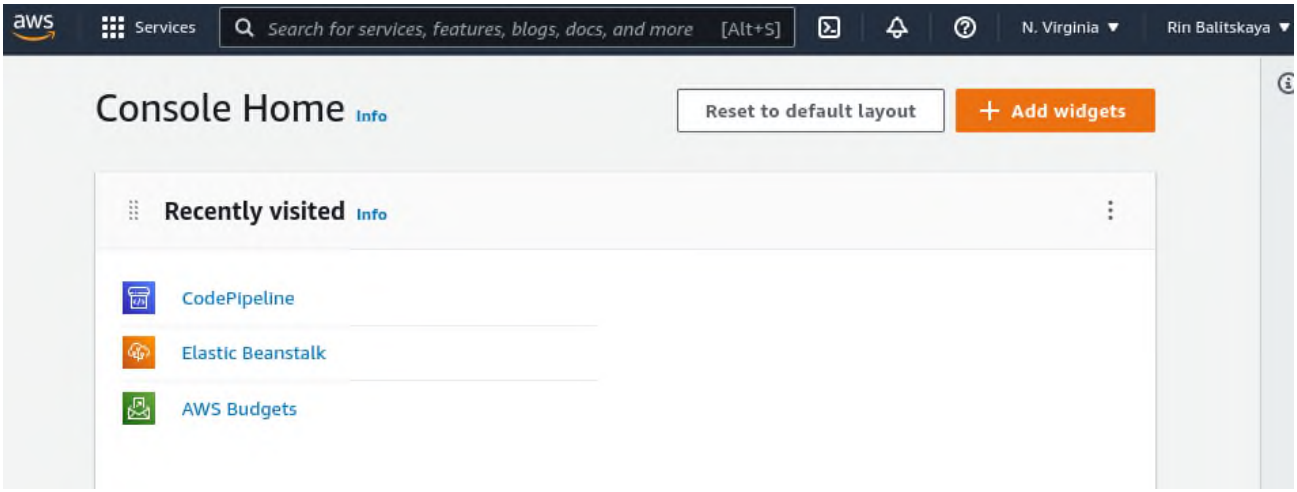


Рис. 3.24. Консоль *AWS*

На відповідній сторінці натиснути «Створити застосунок» (рис. 3.25).

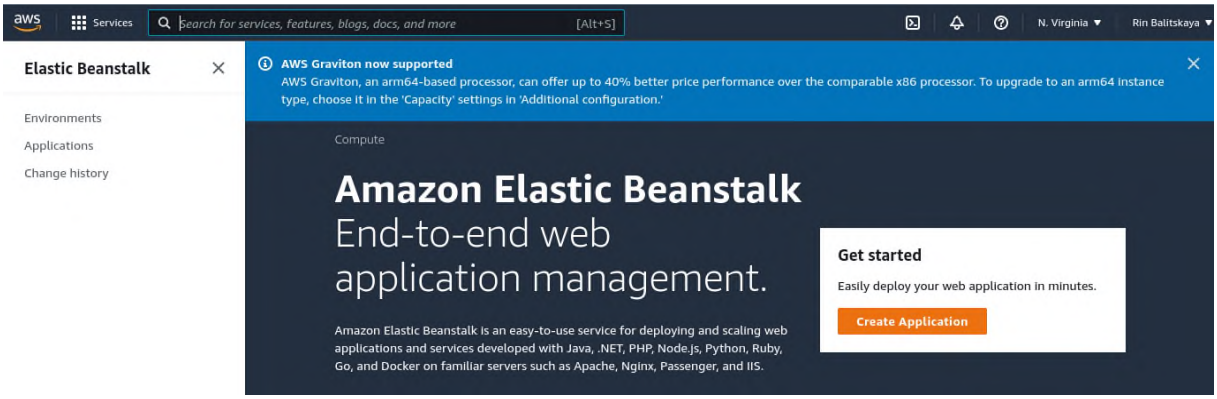


Рис. 3.25. Головна сторінка *Elastic Beanstalk*

У формі створення вказати ім’я застосунку, платформу *Python*, обрати останню доступну платформну гілку, версію платформи (рис. 3.26).

Процес створення застосунку відображається в логах (рис. 3.27).

По завершенню відбувається переадресація на сторінку середовища застосунку (рис. 3.28), де виведено стан здоров’я застосунку, платформа та версія, логи останніх подій.

Сервіс *AWS Elastic Beanstalk* підтримує два способи завантаження коду: напряму з файлу та з репозиторію. Варіант завантаження з репозиторію є більш переважним, оскільки при оновлені коду в репозиторії відбувається автоматичне оновлення застосунку в *AWS*.

Для підключення репозиторію в *AWS* використовується сервіс *CodePipeline*, що також можна знайти в пошуку. На сторінці сервісу слід натиснути на кнопку «Створити пайплайн» (рис. 3.29), та виконати 4 кроки створення.

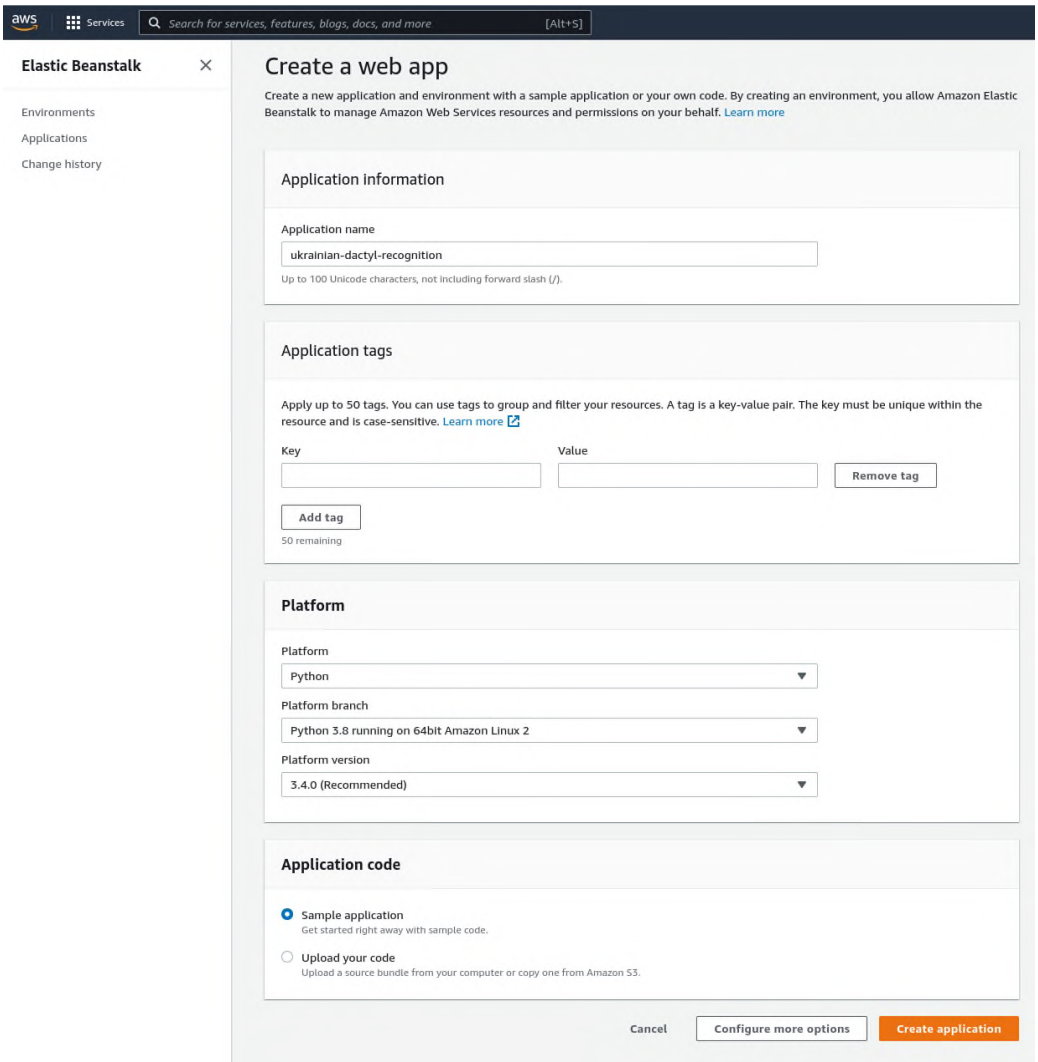


Рис. 3.26. Створення застосунку в *Elastic Beanstalk*

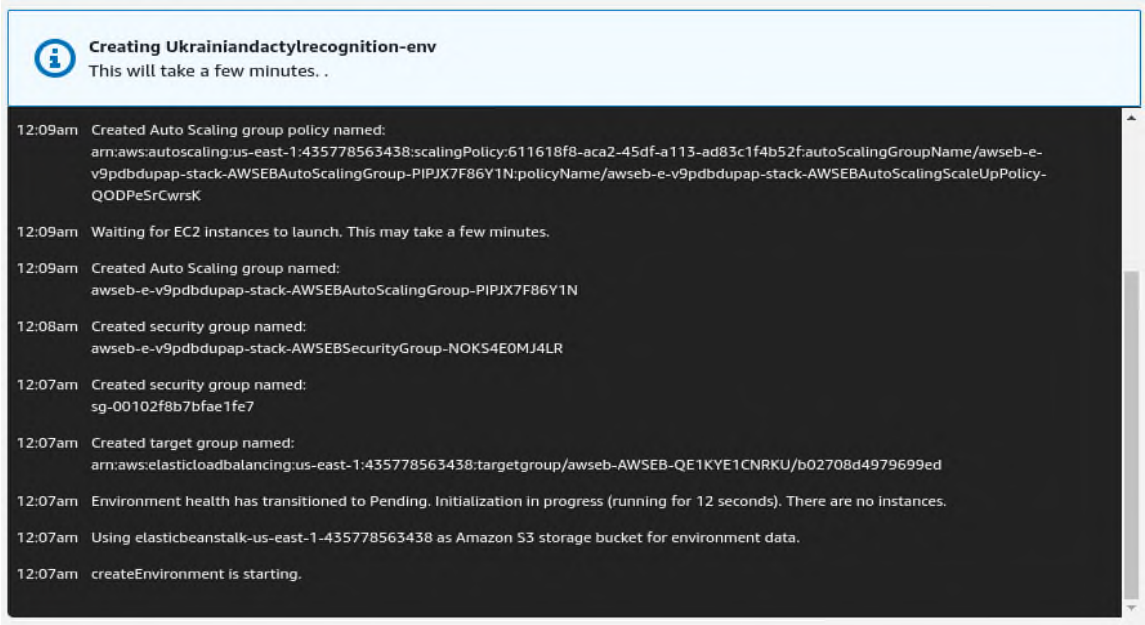


Рис. 3.27. Логи створення застосунку

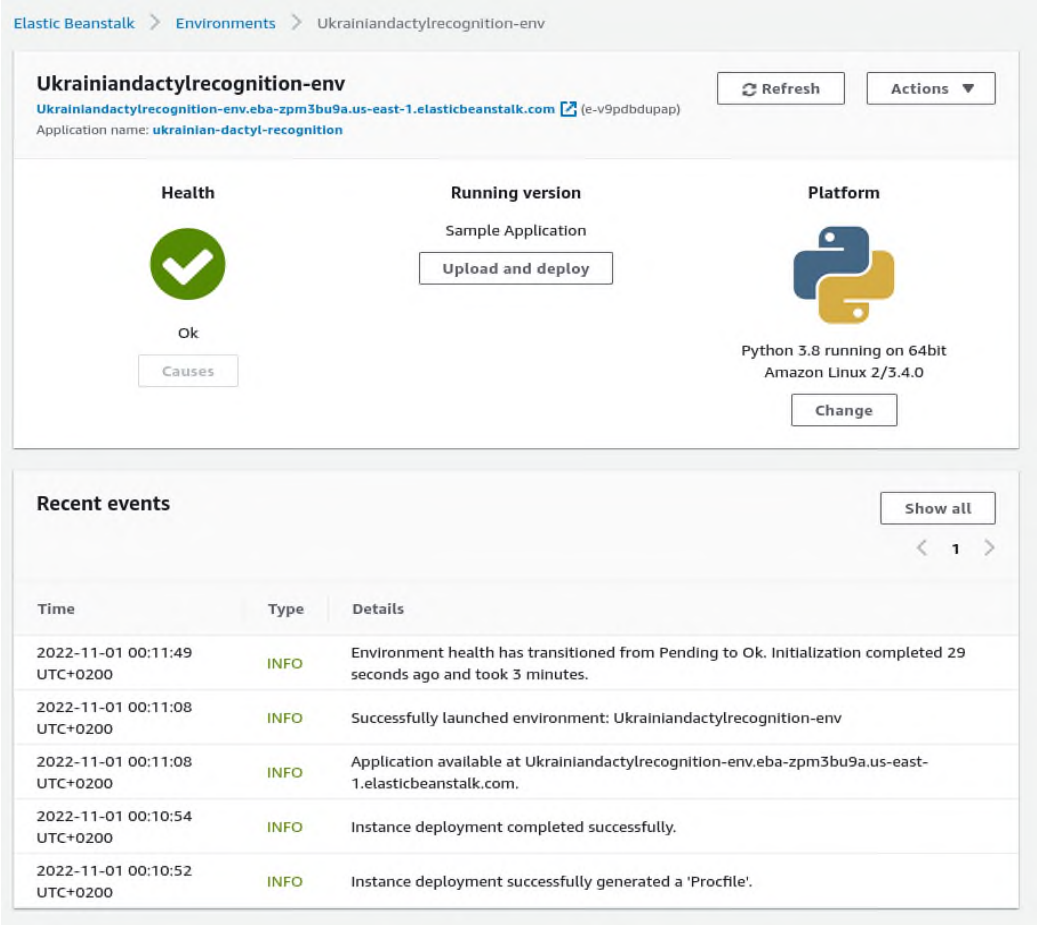


Рис. 3.28. Сторінка середовища застосунку

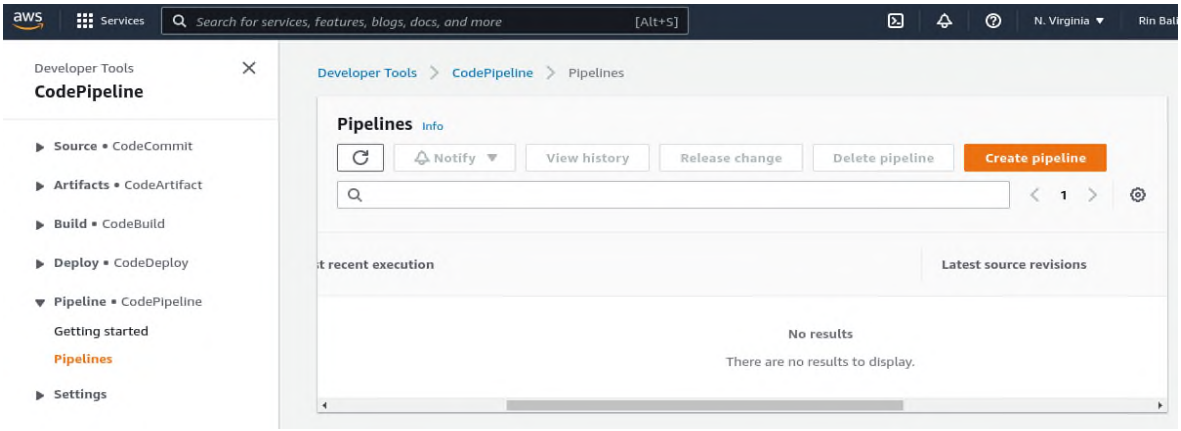


Рис. 3.29. Головна сторінка *CodePipeline*

На першому кроці, у формі створення (рис. 3.30) вказати ім’я пайплайну та натиснути «Далі».

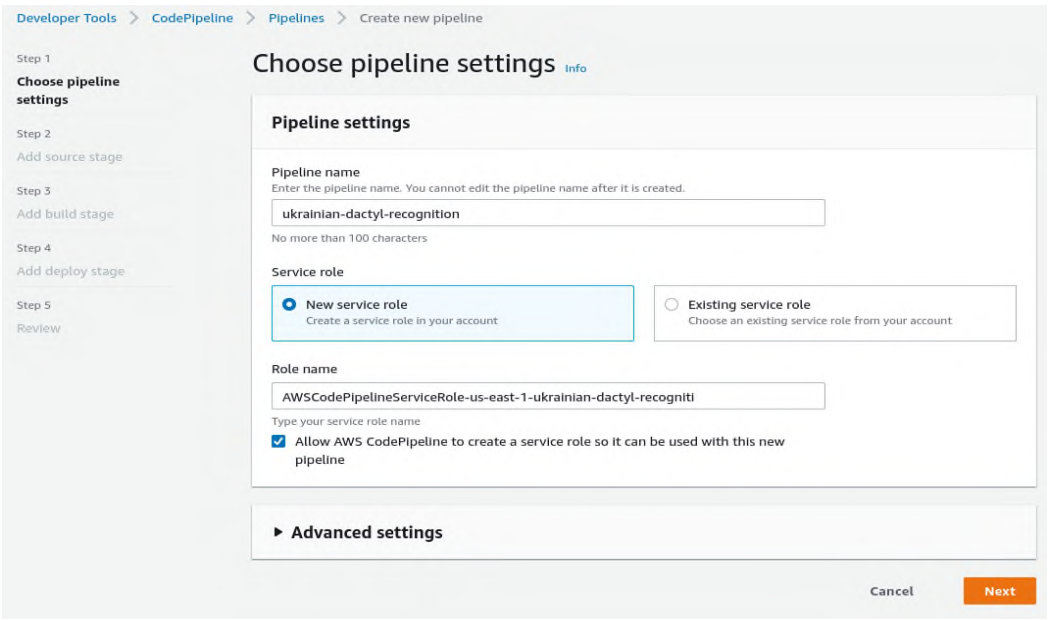


Рис. 3.30. Створення пайплайну

На другому кроці, у випадаючому списку постачальників джерел обрати *GitHub* (*Version 2*), після підгрузки нових полів натиснути «Під’єднатися до *GitHub*» після чого відкриється нове вікно (рис. 3.31), де вказується назва з’єднання та цифровий ідентифікатор облікового запису в *GitHub*. Завершивши

з’єднання, вікно закривається, далі вказується назва репозиторію та основна гілка (рис. 3.32).

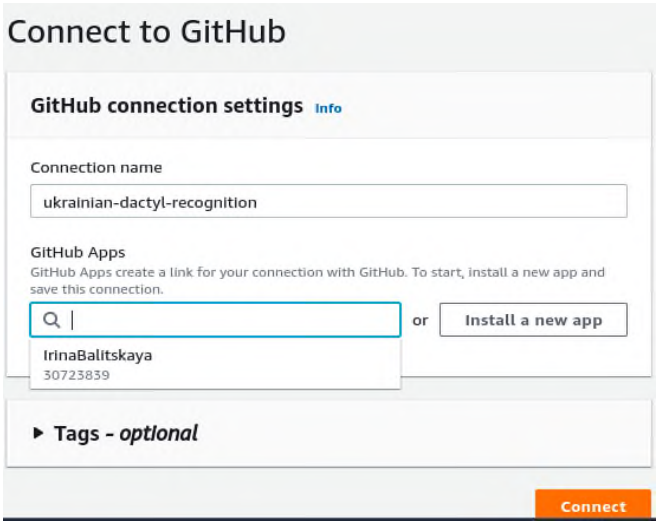


Рис. 3.31. З’єднання з *GitHub*

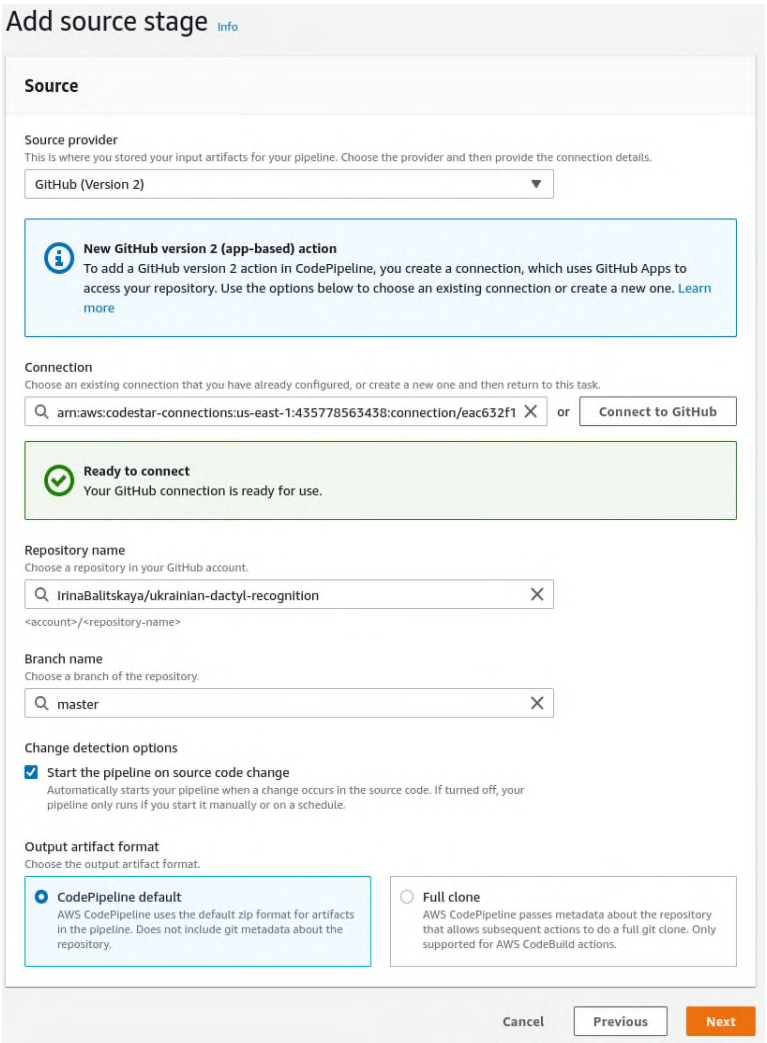


Рис. 3.32. Створення пайплайну

Третій крок – етап збірки (рис. 3.33) – є необов'язковим, його можна пропустити натиснувши на кнопку «Пропустити етап збірки».

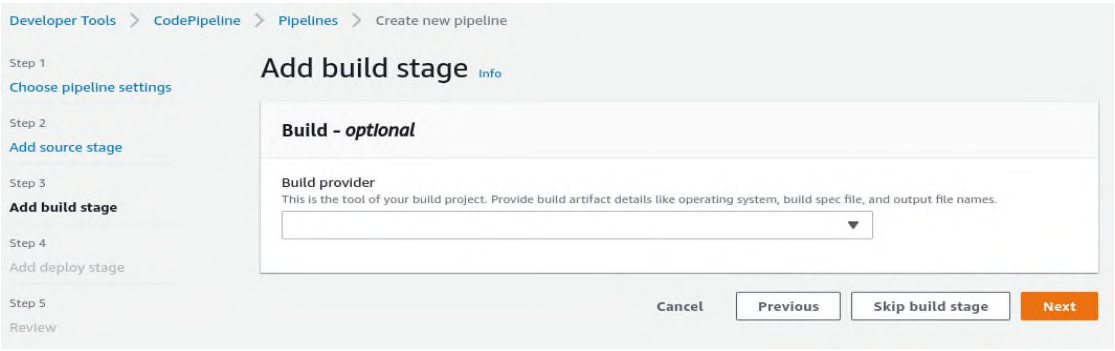


Рис. 3.33. Створення пайплайну

На етапі розгортання (рис. 3.34), останньому кроці, вказується постачальник розгортання – *AWS Elastic Beanstalk*, ім’я застосунку та назва середовища, що автоматично підтягуться із *Elastic Beanstalk*.

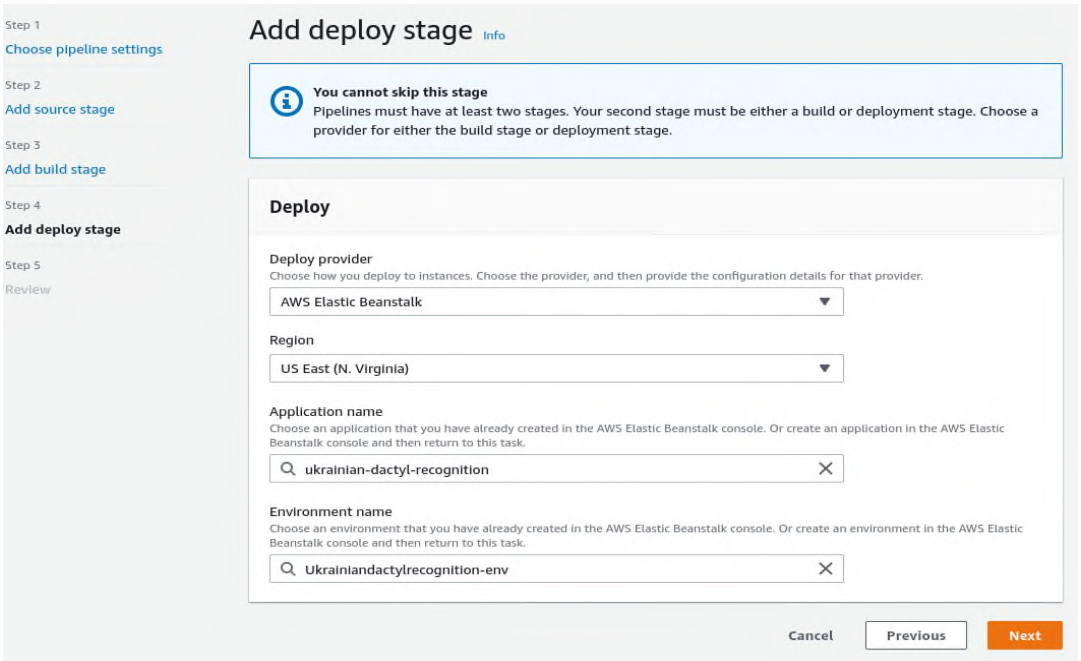


Рис. 3.34. Створення пайплайну

Після проходження всіх кроків відбувається перенаправлення на сторінку (рис. 3.35), де відображається процес розгортання коду з *GitHub* у *Elastic Beanstalk*. Цей процес займає приблизно 5 хвилин.

По завершенню потрібно повернутися до *Elastic Beanstalk*. В середовищах можна побачити список середовищ, що наразі складається із одного застосунку. Список є таблицею, та складається із наступних полів: Назва середовища, Стан здоров'я, Назва застосунку, Дата створення, Востаннє змінено, *URL*, Запущені версії, Платформа, Стан платформи, Назва рівня (рис. 3.36).

В полі *URL* розташовано посилання на сам застосунок (рис. 3.37). Поточне посилання:
http://ukrainiandactylrecognition-env.eba-zpm3bu9a.us-east-1.elasticbeanstalk.com/.

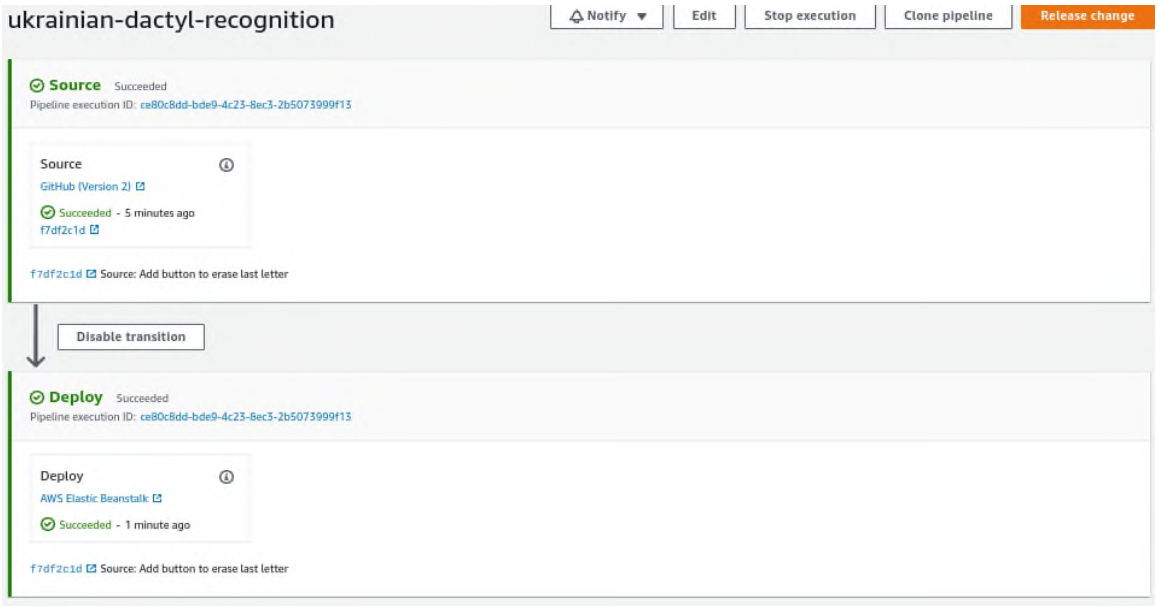


Рис. 3.35. Процес розгортання коду

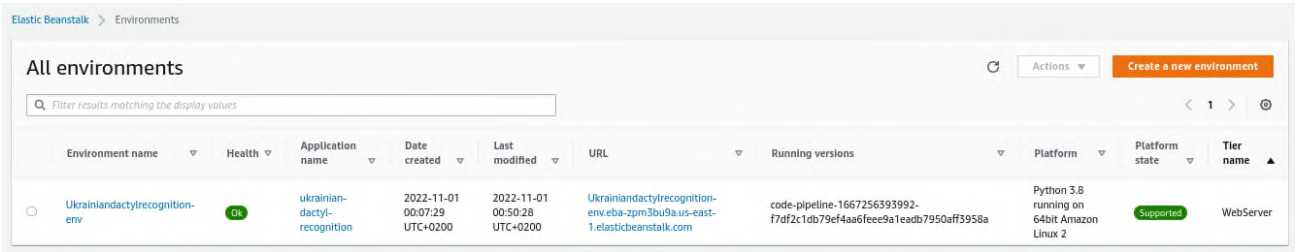


Рис. 3.36. Створене середовище застосунку

Рис. 3.37. Розгорнутий застосунок

3.10. Тестування застосунку

Для тестування застосунку було задіяно 4 особи, кожна з них показує жестами 3 слова: «удав», «ґедзь» та «собака».

Перша особа:

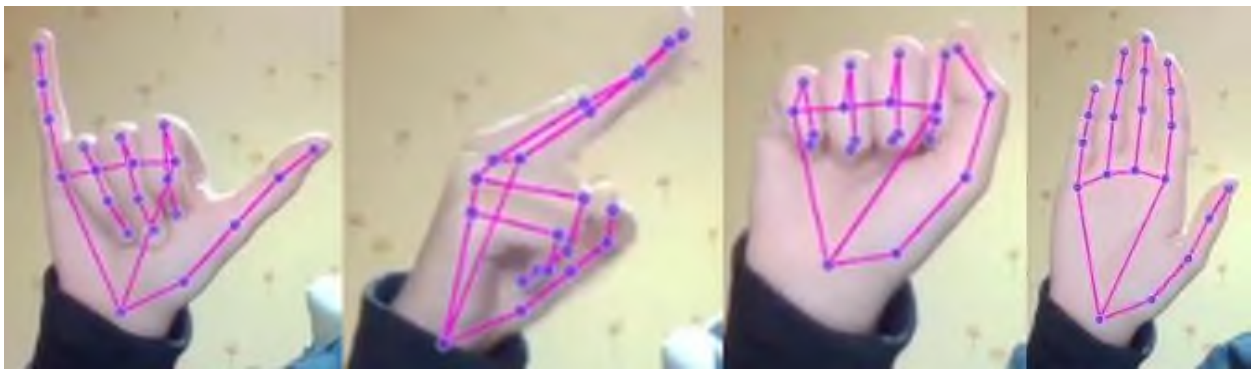


Рис. 3.38. Слово «удав»

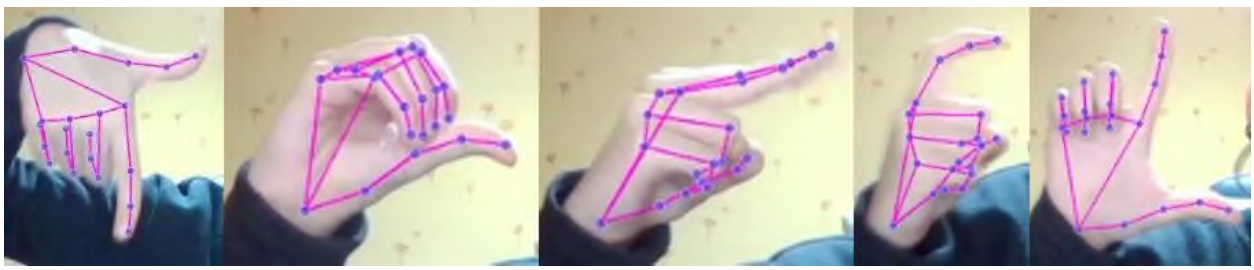


Рис. 3.39. Слово «ґедзь»

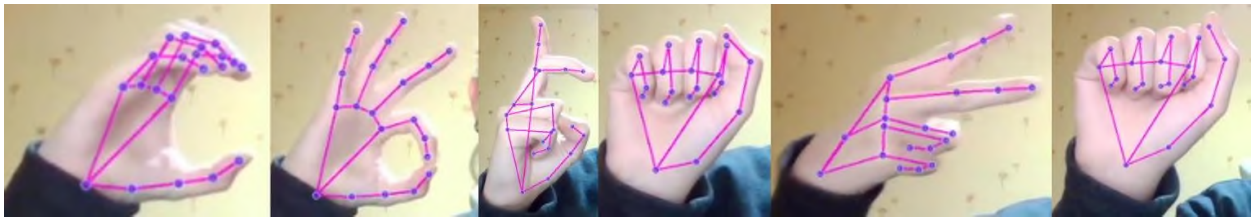


Рис. 3.40. Слово «собака»

Оскільки першою особою є людина, на чиїх руках навчалася нейронна мережа, всі жести було класифіковано правильно.

Друга особа:

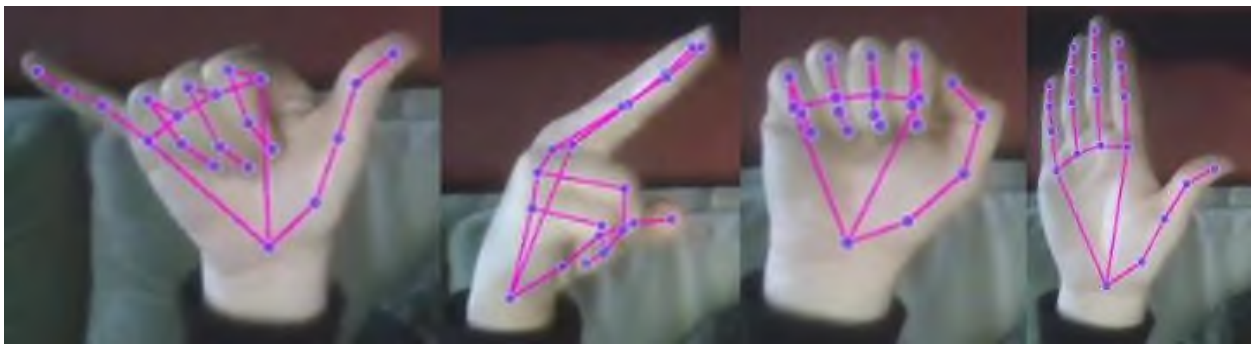


Рис. 3.41. Слово «удав»

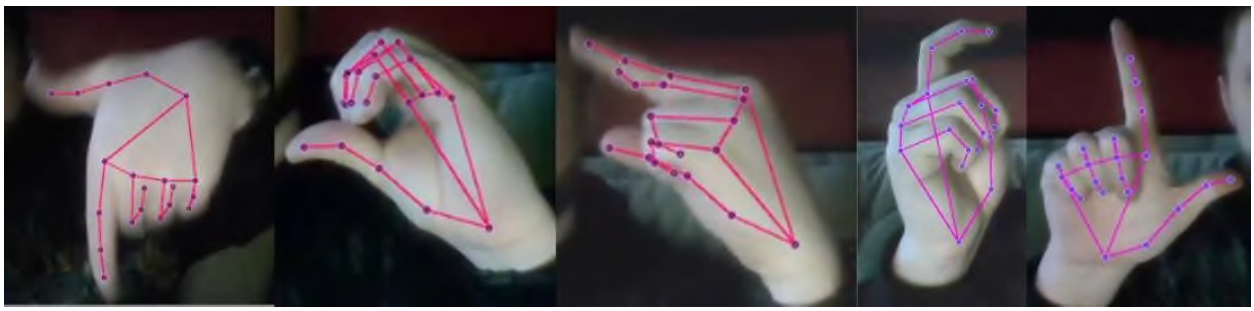


Рис. 3.42. Слово «ґедзь»

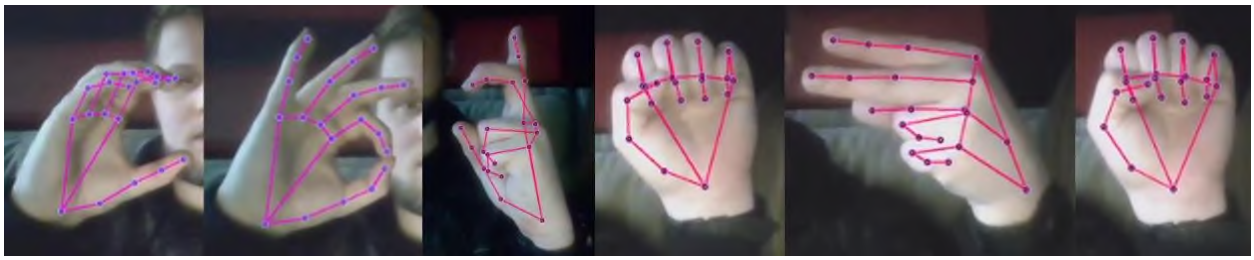


Рис. 3.43. Слово «собака»

У другої особи було неправильно класифіковано жест «З» у слові «ґедзь», жест «Б» у слові «собака».

Третя особа:

Рис. 3.44. Слово «удав»

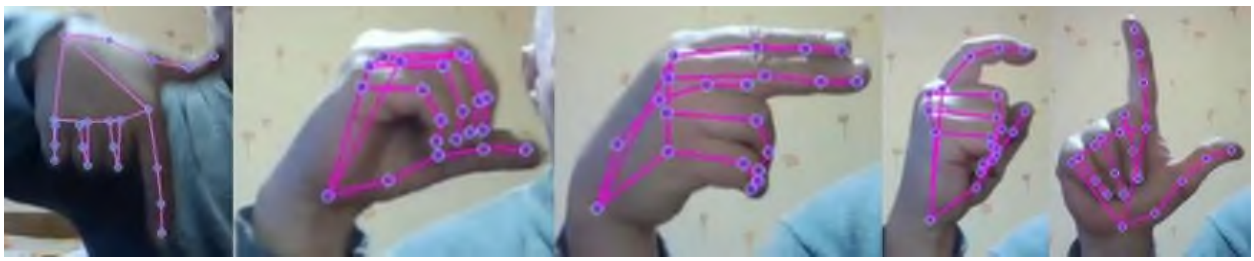


Рис. 3.45. Слово «ґедзь»

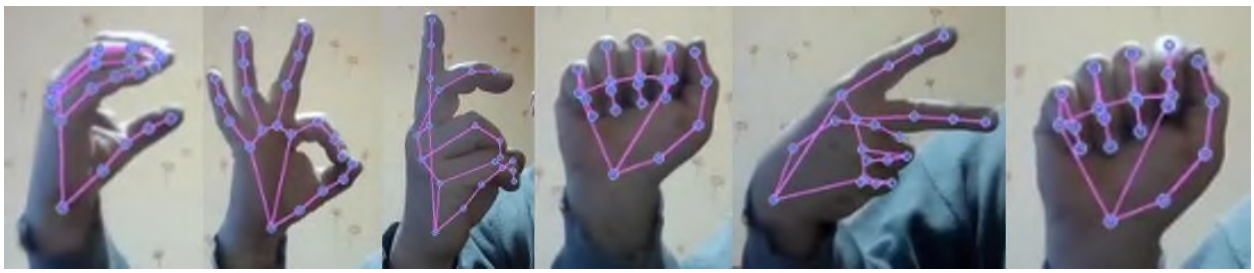


Рис. 3.46. Слово «собака»

У третьої особи невірно класифіковано жести «Д» в слові «удав» та «ґедзь» і «Б» у слові «собака».

Четверта особа:

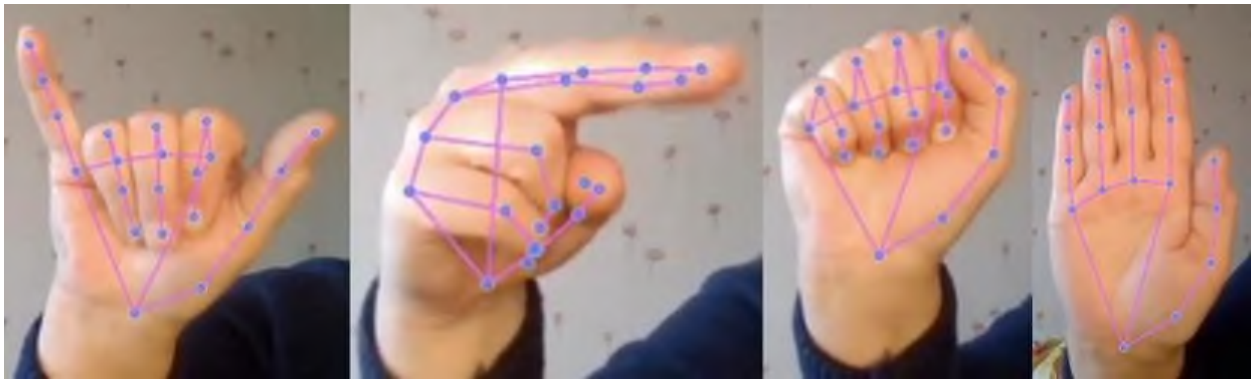


Рис. 3.47. Слово «удав»

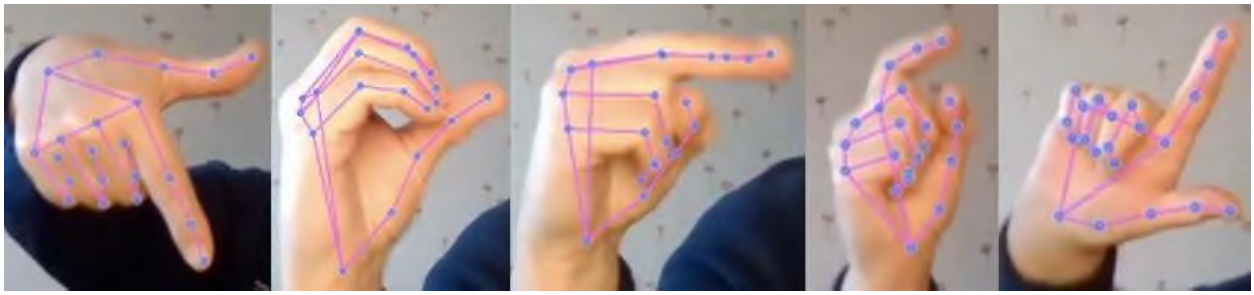


Рис. 3.48. Слово «ґедзь»

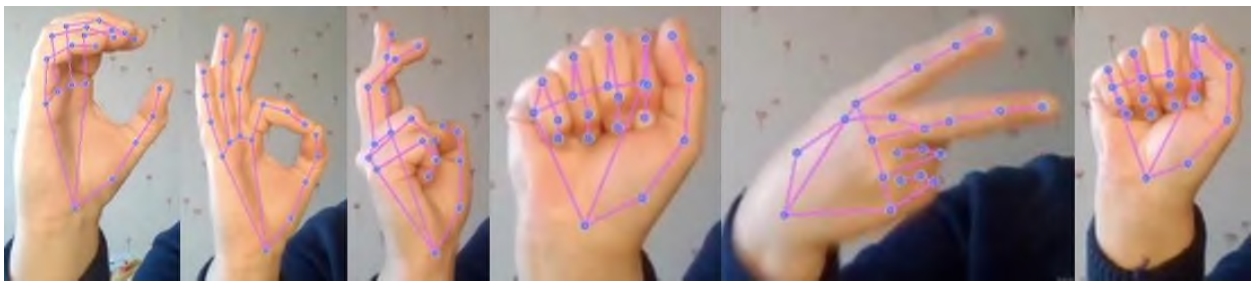


Рис. 3.49. Слово «собака»

Серед жестів четвертої особи помилковий – «З» у слові «ґедзь».

В цілому, тестування можна вважати успішним, із шестидесяти жестів було невірно класифіковано лише шість, причому жести людини, чиї руки фігурували у датасеті, були всі вірно розпізнані.

Помилковий переклад пов’язаний з тим, що деякі жести схожі між собою, наприклад «З» і «Х». Цю неточність можна виправити збільшивши кількість

навчальних зображень та залучивши до навчання інших людей, оскільки кожна людина зображує один і той же жест по-іншому.

3.11. Висновки до розділу

У даному розділі було виконано розробку та тестування застосунку класифікації статичних та динамічних дактилем.

Було визначено інтерпретатор – *Python 3.10* та віртуальне оточення – *Pipenv*, перераховано основні залежності проєкту, а також розгянуто файли проєкту.

Описано процес підготовки даних та створення датасету. Для навчання нейронної мережі виділено 80% створеного датасету, для тестування – 20%.

Розроблено та навчено ШНМ. Для цього було використано фреймворк *Keras*, з якого було імпортовано *Sequential* модель нейронної мережі, шари *LSTM* та *Dense*. В навчанні використано функцію втрат *categorical_crossentropy*, оптимізатор *Adam*, кількість епох рівна 275. Для оцінки навченної нейромережі задіяно функції бібліотеки *sklearn multilabel_confusion_matrix*, в якій імплементовано матрицю плутанини, також *accuracy_score*, яка перевіряє точність навченної мережі на тестовій частині набору даних. Графіки точності та втрат, що залежать від епохи, побудовано за допомогою *TensorBoard*. В кінці навчання виведено кількість параметрів шарів ШНМ та їх загальну кількість, наведено формули, що розраховують кількість параметрів для кожного типу шару. За 275 епох навчання на датасеті розміром 1632 було досягнуто точності нейромережі 88.5%.

Представлено процес класифікації дактилем із веб-камери.

Розглянуто процес розробки серверної частини застосунку, описано її роути.

Розроблено веб-сторінку, описано процес її розмітки, додавання до неї стилів, а також наведено асинхронні запити *jQuery AJAX*, що використовуються для обміну даних із сервером без оновлення сторінки.

Описано процес ініціалізації локального репозиторію системи контролю версій *Git*, а також створення відділеного репозиторію у системі *GitHub*, та їх зв’язування.

Задіяно засоби веб-сервісу *AWS* для розгортання застосунку. Було налаштовано та використано сервіс *Elastic Beanstalk* для деплою застосунку, сервіс *CodePipeline* – для автоматичної підгрузки коду застосунку із *GitHub*.

Проведено тестування додатку, для цього було задіяно 4 особи, жестами їх рук було відображено три слова: «удав», «ґедзь» та «собака». Результат показав, що було коректно класифіковано 54 жести із 60. Статичні та динамічні жести класифікуються однаково добре, що свідчить про те, що запропонована структура нейронної мережі була підібрана правильно. Покращити результат можливо перенавчивши ШНМ на наборі даних, що доповнений жестами рук інших людей і є більшим за обсягом.

ВИСНОВОК

У даній кваліфікаційній роботі було розроблено нейромережевий застосунок класифікації статичних та динамічних дактилем української жестової мови із використанням мови програмування *Python*, фреймворку глибинного навчання *Keras*, бібліотеки комп'ютерного зору *OpenCV*.

Було проведено аналіз предметної області, а саме:

- Розглянуто питання щодо класифікації дактильного алфавіту та методів її реалізації.
- Опрацьовано публікації вітчизняних та зарубіжних авторів з досліджуваної проблематики, проаналізовано існуючі тематичні застосунки.
- Розглянуто найбільш відомі види нейронних мереж, для даної роботи обрано нейромережу *LSTM* – різновид рекурентної нейронної мережі, що здатна навчатися довготривалим залежностям.
- Визначено мету застосування комп'ютерного зору у задачах касифікації жестів.
- Проаналізовано переваги та недоліки бібліотек глибинного навчання мови *Pyhon*, для реалізації застосунку було обрано бібліотеку *Keras*.

Було виконано проєктування застосунку класифікації статичних та динамічних дактилем:

- Розглянуто структуру додатку: модуль набору даних, модуль ШНМ та модуль класифікації дактилем.
- Представлено подання даних, якими буде оперувати нейронна мережа.
- Описано архітектуру нейронної мережі, функції активаці нейронів, оптимізатор та функція втрат.
- Показано процес створення власного набору даних для навчання та тренування нейронної мережі.
- Спроєктовано процес відображення перекладу класифікованої дактилеми: наведено блок-схему алгоритму та графічне відображення процесу.

- Представлено інтерфейс користувача.

Було виконано розробку та тестування застосунку класифікації статичних та динамічних дактилем:

- Визначено інтерпретатор та віртуальне оточення.
- Перераховано основні залежності проєкту, розглянуто файли проєкту.
- Описано процес підготовки даних та створення датасету.
- Розроблено та навчено ШНМ. За 275 епох навчання на датасеті розміром 1632 було досягнуто точності нейромережі 88.5%.
- Представлено процес класифікації дактилем із веб-камери.
- Розглянуто процес розробки серверної частини застосунку.
- Розроблено веб-сторінку.
- Описано процес ініціалізації локального та віддаленого репозиторіїв.
- Задіяно засоби веб-сервісу *AWS* (*Elastic Beanstalk* та *CodePipeline*) для розгортання застосунку.
- Проведено тестування додатку.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Крак Ю.В. Технологія розпізнавання елементів дактильно-жестової мови / Ю.В.Крак, Д.В.Шкільнюк // Штучний інтелект. – 2009. – № 3. – С. 564–572.

2. Давидов М.В., Нікольський Ю.В. Методи та засоби опрацювання зображень реального часу для ідентифікації елементів жестової мови / М.В. Давидов, Ю.В. Нікольський // Штучний інтелект, науково-технічний журнал, Державний університет інформатики і штучного інтелекту. – 2008. – № 1. – С. 131-138.

3. Давидов М.В., Нікольський Ю.В. Пасічник О.В. Дослідження ефективності методів розпізнавання у моделях жестової мови / М.В. Давидов, Ю.В. Нікольський, О.В. Пасічник // Вісн. Нац. ун-ту «Львів. Політехніка». – 2008. – № 621. – С. 117-123.

4. Сіряк Р.В. Модель обробки потокових даних для розпізнавання окремих одиниць жестової мови / Р.В. Сіряк, І.С. Скарга-Бандурова // Вісник Національного технічного університету «ХПІ». Серія: Інформатика та моделювання. – 2018. – № 42. – С. 73-81.

5. *Hlazok O.M., Balytska I.A. A neural network application for classification of a sign language dactylemes // The Tenth World Congress "AVIATION IN THE XXI-st CENTURY – Safety in aviation and space technology", September 28-30, 2022: Proceedings. – K.: NAU, 2022. – Pp. 2.1.5-2.1.8.*

6. *Kasper Kania, Urszula Markowska-KaczmarAmerican: Sign Language Fingerspelling Recognition Using Wide Residual Networks [Electronic resource]. – 2018. – Access mode:https://www.researchgate.net/publication/325073465_American_Sign_Language_Fingerspelling_Recognition_Using_Wide_Residual_Networks (дата звернення 16.11.2022). – Title from the screen.*

7. *Bowen Shi, Aurora Martinez Del Rio: Fingerspelling recognition in the wild with iterative visual attention [Electronic resource]. – 2019. – Access mode:*

https://arxiv.org/pdf/1908.10546.pdf (дата звернення 16.11.2022). – Title from the screen.

8. *Byeongkeun Kang, Subarna Tripathi, Truong Q. Nguyen Real-time: Sign Language Fingerspelling Recognition using Convolutional Neural Networks from Depth map [Electronic resource]. – 2015. – Access mode: https://arxiv.org/pdf/1509.03001.pdf (дата звернення 16.11.2022). – Title from the screen.*

9. *Amrita Thakur, Pujan Budhathoki: Real Time Sign Language Recognition and Speech Generation [Electronic resource]. – 2020. – Access mode: https://www.irojournals.com/iroiip/V2/I2/01.pdf (дата звернення 16.11.2022). – Title from the screen.*

10. *What are neural networks? [Electronic resource] – Access mode: https://www.ibm.com/cloud/learn/neural-networks – Title from the screen.*

11. *What are convolutional neural networks? [Electronic resource] – Access mode: https://www.ibm.com/cloud/learn/convolutional-neural-networks – Title from the screen.*

12. *What are recurrent neural networks? [Electronic resource] – Access mode: https://www.ibm.com/cloud/learn/recurrent-neural-networks – Title from the screen.*

13. *Sepp Hochreiter, Jürgen Schmidhuber: Long short-term memory [Electronic resource]. – 1997. – Access mode: https://www.bioinf.jku.at/publications/older/2604.pdf (дата звернення 16.11.2022). – Title from the screen.*

14. *Long Short-Term Memory (LSTM) [Electronic resource] – Access mode: https://www.mathworks.com/discovery/lstm.html – Title from the screen.*

15. *A Guide to Recurrent Neural Networks: Understanding RNN and LSTM Networks [Electronic resource] – Access mode: https://builtin.com/data-science/recurrent-neural-networks-and-lstm – Title from the screen.*

16. *What is OpenCV? The Complete Guide [Electronic resource] – Access mode: https://viso.ai/computer-vision/opencv/ – Title from the screen.*

17. *Computer Vision What it is and why it matters [Electronic resource] – Access mode: https://www.sas.com/en_us/insights/analytics/computer-vision.html – Title from the screen.*

18. *Pytorch documentation [Electronic resource] – Access mode: https://pytorch.org/docs/stable/index.html – Title from the screen.*

19. *TensorFlow [Electronic resource] – Access mode: https://www.tensorflow.org/ – Title from the screen.*

20. *About Keras [Electronic resource] – Access mode: https://keras.io/about/ – Title from the screen.*

21. *Keras vs TensorFlow vs PyTorch: Comparison of the Deep Learning Frameworks [Electronic resource] – Access mode: https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch – Title from the screen.*

22. *Keras vs Tensorflow vs Pytorch: Key Differences Among the Deep Learning Framework [Electronic resource] – Access mode: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article – Title from the screen.*

23. *What is Python? Executive Summary [Electronic resource] – Access mode: https://www.python.org/doc/essays/blurb/ – Title from the screen.*

24. *PyCharm [Electronic resource] – Access mode: https://www.jetbrains.com/help/pycharm/quick-start-guide.html – Title from the screen.*

25. *OpenCV [Electronic resource] – Access mode: https://opencv.org/about/ – Title from the screen.*

26. *Python – Facial and hand recognition using MediaPipe Holistic [Electronic resource] – Access mode: https://www.geeksforgeeks.org/python-facial-and-hand-recognition-using-mediapipe-holistic – Title from the screen.*

27. *What is NumPy? [Electronic resource] – Access mode: https://numpy.org/doc/stable/user/whatisnumpy.html – Title from the screen.*

28. *What is Flask Python [Electronic resource] – Access mode: https://pythonbasics.org/what-is-flask-python/ – Title from the screen.*

29. *What Is HTML? Hypertext Markup Language Basics Explained [Electronic resource] – Access mode: https://www.hostinger.com/tutorials/what-is-html – Title from the screen.*

30. *What Is CSS and How Does It Work? [Electronic resource] – Access mode: https://www.hostinger.com/tutorials/what-is-css – Title from the screen.*

31. *What is jQuery? [Electronic resource] – Access mode: https://jquery.com/ – Title from the screen.*

32. *jQuery - AJAX Introduction [Electronic resource] – Access mode: https://www.w3schools.com/jquery/jquery_ajax_intro.asp – Title from the screen.*

33. *Cloud computing with AWS [Electronic resource] – Access mode: https://aws.amazon.com/ru/what-is-aws/ – Title from the screen.*

34. *What is AWS Elastic Beanstalk? [Electronic resource] – Access mode: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html – Title from the screen.*

35. *AWS CodePipeline FAQs [Electronic resource] – Access mode: https://aws.amazon.com/codepipeline/faqs – Title from the screen.*

36. *MediaPipe Hands [Electronic resource] – Access mode: https://google.github.io/mediapipe/solutions/hands.html – Title from the screen.*

37. *The Sequential model [Electronic resource] – Access mode: https://keras.io/guides/sequential_model/ – Title from the screen.*

38. *LSTM layer [Electronic resource] – Access mode: https://keras.io/api/layers/recurrent_layers/lstm/ – Title from the screen.*

39. *Dense layer [Electronic resource] – Access mode: https://keras.io/api/layers/core_layers/dense/ – Title from the screen.*

40. *Activation Functions in Neural Networks [Electronic resource] – Access mode: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6/ – Title from the screen.*

41. *Softmax Activation Function — How It Actually Works [Electronic resource] – Access mode https://towardsdatascience.com/softmax-activation-function-how-it-actually-works-d292d335bd78 – Title from the screen.*

42. *Adam [Electronic resource] – Access mode: https://keras.io/api/layers/core_layers/dense/ – Title from the screen.*

43. *Diederik P. Kingma, Jimmy Ba: Adam: A Method for Stochastic Optimization [Electronic resource]. – 2017. – Access mode: https://arxiv.org/abs/1412.6980 (дата звернення 16.11.2022). – Title from the screen.*