Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Інтелектуальний парсер для перетворення макетів web-сторінок у HTML розмітку
(тема)

Виконав:
студент 2 курсу, групи СШІм-20-3
Коваль А.Ф.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник доц. Золотухін О.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«» _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Коваль Анні Федорівні
(прізвище, ім'я, по батькові)

1. Тема роботи Інтелектуальний парсер для перетворення макетів web-сторінок у HTML розмітку

затверджена наказом університету від 24 березня 20 22 р. № 414 Ст

2. Термін подання студентом роботи до екзаменаційної комісії __ _____ 20 ___ р.

3. Вихідні дані до роботи Науково-технічні публікації, дані Інтернет-джерел та наукових проектів щодо розробки та дослідження алгоритмів для парсингу макетів web-сторінок у HTML розмітку за допомогою штучного інтелекту

4. Перелік питань, що потрібно опрацювати в роботі

1. Аналіз предметної галузі та постановка задачі

2 Вибір та обгрунтування алгоритмів для парсингу макетів web-сторінок у HTML розмітку за допомогою штучного інтелекту, архітектури та його модифікацій для програмної реалізації, технології для програмної реалізації

3 Проведення парсингу макетів web-сторінок у HTML розмітку

4 Впровадження результатів та перспективи розвитку

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Рисунок 3.1 – Розмітка в пам'яті нейронної мережі, Рисунок 3.2 – Відображення моделі послідовності, Рисунок 3.3 – Варіанти прогнозування рішення, Рисунок 3.4 – Основні розділи побудови HTML документа, Рисунок 3.5 – Основні моделі нейронних мереж застосовуваних при створенні розмітки, Рисунок 3.6 – Версія процесу для кожного тимчасового кроку в LSTM, Рисунок 3.7 – Програмний код алгоритму передбачення з використанням LSTM, Рисунок 3.8 – Процес тренування нейронної мережі, Рисунок 3.9 – Приклад зображення для генерації коду, Рисунок 3.10 – Результат роботи алгоритму у вигляді HTML-коду, Рисунок 3.11 – Результат роботи алгоритму у вигляді візуалізованого HTML-коду

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	28.03.2022	Виконано
2	Аналіз предметної галузі	29.03.2022	Виконано
3	Постановка задачі	30.03.2022	Виконано
4	Аналіз методів пошуку парсингу макетів web-сторінок	05.04.2022	Виконано
	у HTML розмітку		
5	Виявлення характеристик парсингу макетів	10.04.2022	Виконано
	web-сторінок у HTML розмітку		
6	Вибір технологій для програмної реалізації	12.04.2022	Виконано
7	Програмна реалізація моделі	13.04.2022	Виконано
8	Оформлення пояснювальної записки	20.04.2022	Виконано
9	Надання пояснювальної записки на перевірку керівнику	26.04.2022	Виконано
10	Захист кваліфікаційної роботи		Виконано

Дата видачі завдання 28 березня 20 22 р.

Студент _____
(підпис)

Керівник роботи _____ доц Золотухін О.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 60 с., 11 рис., 3 дод., 19 джерел.

ВИБІРКА, ДОВГА КОРОТКОСТРОКОВА ПАМ’ЯТЬ, ЕНКОДЕР, НЕЙРОННА МЕРЕЖА, ОЗНАКИ ЗОБРАЖЕННЯ, РЕКУРЕНТНА НЕЙРОННА МЕРЕЖА.

Об'єкт дослідження – сучасні методи інтелектуального парсингу макетів web-сторінок у HTML розмітку.

Предмет дослідження – використання нейронних мереж для інтелектуального парсингу макетів web-сторінок у HTML розмітку.

Мета роботи – пошук найбільш оптимального за продуктивністю алгоритму для парсингу макетів web-сторінок у HTML розмітку за допомогою глибинної нейронної мережі.

У результаті роботи був проведений аналіз технічної літератури, предметної галузі, виявлено проблемні місця та поставлено задачу розробки готового рішення. Були сформовані вимоги та досліджено методи автоматичної розробки веб-сайтів. На основі отриманих результатів розроблено модель, що за допомогою алгоритму дозволяє моделі достатньо точно передбачити веб-компоненти.

ABSTRACT

Explanatory note: 60 p., 11 fig., 3 ann., 19 sources.

SAMPLING, LONG SHORT-TERM MEMORY, ENCODER, NEURAL NETWORK, SIGNS OF IMAGE, RECURRENT NEURAL NETWORK.

The object of research – modern methods of intelligent parsing of web-page layouts in HTML markup.

The subject of research is the use of neural networks for intelligent parsing of web-page layouts in HTML markup.

The aim of the work is to find the most optimal algorithm for parsing web page layouts in HTML markup using a deep neural network.

As a result of the work the analysis of technical literature, subject branch was carried out, problem places were revealed and the task of development of the ready decision was set. Requirements were formed and methods of automatic development of web sites were investigated. Based on the obtained results, a model was developed that, with the help of the algorithm, allows the model to predict web components quite accurately.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

ПЗ − програмне забезпечення;

ШІ − Artificial intelligence, AI − штучний інтелект – здатність інженерної системи обробляти, застосовувати та вдосконалювати здобуті знання та вміння;

ANN – Artificial Neural Networks – нейронні мережі, обчислювальні системи, натхнені біологічними нейронними мережами, що складають мозок тварин;

API – Application Programming Interface – це набір публічних методів та властивостей, які використовуються для взаємодії з іншими об’єктами у програмі;

CSS – Cascading Style Sheets − це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду;

DL – Deep Learning − глибоке навчання, підмножина машинного навчання, що ґрунтується на наборі алгоритмів, які намагаються моделювати високорівневі абстракції в даних, застосовуючи глибинний граф із декількома обробними шарами, що побудовано з кількох лінійних або нелінійних перетворень;

DOM – Document Object Model – специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами;

Frontend – це публічна частина web-додатків (веб-сайтів), з якою користувач може взаємодіяти і контактувати напряму;

GUI – Graphical user interface − тип інтерфейсу, який дає змогу користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки;

HTML − HyperText Markup Language − стандартизована мова розмітки документів для перегляду веб-сторінок у браузері;

LSTM − Long short-term memory − особливий різновид архітектури рекурентних нейронних мереж, здатна до навчання довгострокових залежностей;

RNN − Recurrent Neural Networks − це мережі, що містять зворотні зв'язки та дозволяють зберігати інформацію.

ВСТУП

Сьогодні максимальна кількість нинішніх сфер людської діяльності активно співіснують із сучасними технологіями та змінюються швидкими темпами, і процес front-end розробки звичайно також підпадає під ці зміни. Веб-розробка інтерфейсу пройшла довгий шлях, від статичних до динамічних веб-сайтів та використання чат-ботів. Навіть якщо під час розробки інтерфейсу відбувається багато технологічних змін, це неможливо повністю автоматизувати. Фронт-енд розробка вимагає обережного використання різноманітних інструментів або платформ кваліфікованими розробниками для створення привабливого та привабливого веб-сайту. Важливість AI для front-end розробки полягає в тому, що AI можна використовувати для автоматизації повторюваних завдань, які виникають як частина циклу front-end розробки. Використання AI зменшить навантаження на розробників, підвищуючи їх продуктивність.

Поряд з розробкою веб-сайтів, на пошук і виправлення помилок витрачається стільки ж або більше часу. Автоматизація процесу тестування за допомогою штучного інтелекту може призвести до дуже необхідних змін у галузі. AI можна використовувати для пошуку успішних методів тестування, швидкого виправлення помилок, тим самим підвищуючи стандарт розробки. AI можна використовувати навіть для виявлення помилок, які жоден інший компілятор не може виявити.

Користувальницький досвід є найважливішим фактором, який кожен розробник інтерфейсу має мати на увазі під час розробки веб-сторінки. На сьогодні існує наявність чату в реальному часі майже на всіх веб-сайтах. У момент, коли ви задасте будь-які запитання, ви отримаєте рішення для нього в той же момент. Чат-боти були інтегровані у веб-сайти для

покращення досвіду клієнтів на сайті. Чат-боти були корисними для того, щоб зробити процес спілкування більш комфортним. Машини можуть навіть розшифровувати людські емоції, завдяки чому спілкування виглядає природним. Користувачам буде легше взаємодіяти з веб-сайтами в найближчі роки без будь-яких перешкод [11].

Машинне навчання − це вивчення різних даних і подальше використання їх навчання для прийняття важливих рішень. Машини повинні використовувати коди, які вже написані розробниками, з метою навчання. Розробникам завжди доведеться спочатку надавати вхідні дані, щоб моделі могли вчитися на цьому. Інструменти штучного інтелекту можуть створювати прототипи HTML, але здатність таких інструментів створювати анімацію та інтерактивність на стороні клієнта залишається під питанням. Статичні речі, які можна розробити за допомогою HTML і CSS, можна зробити за допомогою технологій AI. Крім того, розробка інтерфейсу не обмежується лише створенням інтерфейсу користувача. Попереду ще багато роботи, зв’язок з API, побудова моделі тощо потребує досвіду кваліфікованого розробника [6].

Інновації відбуваються, а технології змінюються високими темпами. Оскільки штучний інтелект стає все більш популярним і поширеним, очікується, що ці технології знайдуть застосування в кожному аспекті нашого життя. Безсумнівно, що AI може полегшити життя фронтенд-розробників, автоматизуючи повторювані завдання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка задачі

Створення інтуїтивно зрозумілого та привабливого веб-сайту є основною метою веб-розробників. Їм потрібно пройти кілька процесів, включаючи малювання концептуальних ескізів, створення прототипів і тестування веб-сайту, перш ніж запустити його у використання. Ці процеси не відбуваються за одну ніч. Розробники витрачають тижні й місяці на створення красивого та адаптивного веб-сайту. Але розвиток технологій полегшує їм справу.

Сучасні технології, такі як штучний інтелект (AI) і машинне навчання, прискорюють розробку інтерфейсу, а також роблять кодування та тестування макета веб-сайту простішим, швидшим та ефективнішим. Особливо глибоке навчання, частина машинного навчання, відіграє вирішальну роль у розробці інтерфейсу.

Машинне навчання, галузь Штучного Інтелекту, пропонує ще одну перевагу у взаємодії людини та машини. Не маючи можливостей навчання, програми будуть підходити до проблеми однаково час від часу і робити ту саму помилку, не змінюючи чи оптимізуючи рішення на основі попереднього досвіду.

Але, машинне навчання – це технологія, яка дозволяє веб-додаткам адаптуватися з часом, спостерігаючи та вивчаючи звички, особливості та переваги користувачів. Завдяки тому, що програми стають розумнішими, покращується досвід користувача.

Завдяки вищезгаданим конкурентним перевагам, чому веб-сайти з підтримкою штучного інтелекту на сьогоднішній день не розгорнуті

скрізь? Одна з причин полягає в тому, що, незважаючи на свою довгу історію, AI все ще залишається технологією, що розвивається, що стосується основних інформаційних технологій.

Глибоке навчання − це підкатегорія машинного навчання, яка включає навчання та умовиводи. Існують штучні нейронні мережі, які потрібно навчити. Користувачам потрібно буде передати набори даних у ненавчену нейронну мережу разом із правильними відповідями. Це означає, що навчання включатиме велику кількість вхідних даних і відповідних результатів.

Після навчання нейронна мережа стане здатною аналізувати та розуміти вхідні дані, а потім відповідним чином повертати вихідні дані. Насамперед, моделі глибокого навчання використовуються для кількох випадків використання ШІ, як-от, наприклад, комп’ютерний зір та обробка природної мови [12].

Розробники інтерфейсу можуть використовувати подібні моделі для проектування елементів інтерфейсу користувача, навчаючи нейронні мережі з відповідними даними.

1.2 Класифікація обраної задачі

Розглянемо основні задачі створення програмної системи, що потрібно вирішити:

− створення програмної моделі інтелектуального парсеру для перетворення макетів web-сторінок у HTML розмітку;

− отримання програмного коду без використання систем керування вмістом;

− перетворення зображення на повноцінний макет веб-сайту;

− можливість створення візуального інтерфейсу веб-версії.

Отже, виходячи з наведених задач і проблем, для виконання завдання розробки програмної моделі інтелектуального парсеру, що застосовується для перетворення макетів web-сторінок у HTML розмітку, треба проаналізувати існуючі рішення для вирішення кожної з оголошених проблем, а потім згрупувати здобуті результати. Після виконання цих процедур повинен з’явитися набір рішень та інструментів для розв'язку задачі з перетворення макетів web-сторінок у HTML розмітку.

2 ОГЛЯД МЕТОДІВ ПАРСИНГУ МАКЕТІВ WEB-СТОРІНОК У HTML РОЗМІТКУ

2.1 Огляд задач перетворення макетів web-сторінок у HTML розмітку

Для впровадження Штучного Інтелекту у фронтенд-автоматизацію вже сьогодні можна застосовувати синтезовані набори даних та алгоритми навчання. У 2014 році команда Grid вперше спробувала розробити платформу для автоматичної генерації веб-додатків, запустивши компанію Kickstarter. Новітня технологія привернула увагу засобів масової інформації. Джеррі Янг, один із перших інвесторів Grid, описав цю платформу, як керовану систему на основі Штучного Інтелекту: «Хмарний ШІ Grid розробить веб-дизайн та створить елегантні веб-сайти, усуваючи ті частини веб-розробки, що віднімають багато часу і є непотрібними» [4]. Але багато користувачів Reddit заявили про свої претензії до компанії, кажучи, що технологія Grid більше схожа на систему, яка додає зображення та застосовує різноманітні кольори до декількох стандартних шаблонів, і не використовує весь потенціал, про який було сказано в рекламній компанії.

Хоча, такі платформи, як Jimdo Dolphin, Bookmark, Wix ADI, Firedrop є більш везучими. Алгоритм, що використовується в основі моделі Штучного Інтелекту в даних платформах, створює веб-додаток з готових зображень, таблиць, шрифтів та інших компонентів. Спочатку платформа задає кілька запитань, коли розробляє ці компоненти, після чого платформа, виходячи з потреб користувача, обирає ті, що найкраще комбінуються із заготовленими зразками.

Від наповнення бази даних залежить спроможність Штучного Інтелекту створювати веб-сайт. На додачу, маємо на увазі, що існує так

званий «простий» набір інструкцій та шаблонів, часто використовуваних в веб-розробці. Системи генерації веб-додатків на базі Штучного Інтелекту мають доступ до великої кількості дизайнів та макетів сайтів. Wix, наприклад, користуються більше 100 мільйонів людей, що без сумніву підтверджує, що це простий, швидкий і ефективний спосіб створення веб-сайту [5].

На додачу, дані платформи здатні застосовувати методи для захисту від злому. Крім цього, системи веб-додатків на базі Штучного Інтелекту можуть вносити корективи або відправляти їх на пошту користувачу, таким чином контролюючи аналітику сайту. Треба зазначити, важливим є A/B тестування на сайті, на базі якого висувається список можливих поліпшень функціональних та візуальних можливостей, від початку встановлених для веб-сайту. В даний час головною проблемою при автоматизації фронтенд розробки є обмежена обчислювальна потужність.

Є два типи даних, які розробники інтерфейсу можуть використовувати для навчання нейронної мережі. Перший − це скріншоти графічного інтерфейсу користувача (GUI), а другий − мокапи від дизайнерів. Скріншоти та мокапи відображають необхідний інтерфейс та дизайн, що повинен бути реалізованим у вигляді коду мовою розмітки.

Такий спосіб навчання зробить нейронну мережу здатною генерувати код із скріншота графічного інтерфейсу. Після аналізу скріншотів і макетів нейронна мережа з’ясує способи кодування цих зображень у HTML або CSS. Для зображень із текстом мережі також пройдуть етапи розпізнавання тексту.

Після цих основних кроків нейронна мережа вступає в дію і використовує свої знання про об’єкти, їх положення та макети дизайну для генерації коду для елементів дизайну. Згодом ця нейронна мережа стає

більш здатною розуміти елементи візуального дизайну та наміри кодування.

Отже, щоб створити нейронну мережу, яка може генерувати розмітку HTML/CSS за певним скриншотом, при навчанні нейронної мережі ми даємо кілька зображень з правильно структурованим HTML. Так система вчиться послідовно передбачати всі використовувані теги, тобто вчиться «читати» зображення. При передбаченні наступних тегів, алгоритм звертається до вже передбачених і складає правильні послідовності тегів згідно наданого і вивчених раніше зображень [19].

Використання вже тренованих мереж для вирішення практичних задач мало відрізняється від навчання моделей з нуля. Текст генерується покроково, супроводжується однаковими картинками. Замість програмування алгоритму правильних тегів HTML, вона отримується вже згенеровану розмітку. Далі — передбачається наступний тег. Передбачення ініціюється «первинним тегом» і закінчується «кінцевим тегом», або досягненням максимального краю [2].

Нейронна мережу вибудовується у три етапи. Для початку береться мінімальна версія, щоб закріпити на ній рухомі частини. На другому етапі (з HTML) автоматизуються всі кроки та нейронна мережа знайомиться з послідовностями. У фінальній ітерації створюється модель, яка узагальнює і досліджує LSTM-шар (довга короткострокова пам'ять).

Хоча енергоспоживання глибокого навчання вище, варіанти використання цієї технології усувають перешкоди у веб-дизайні та роблять весь процес безпроблемним. Оскільки існують мільйони веб-сайтів, розробникам не потрібно турбуватися про дані, необхідні як вхідні дані для навчання моделей глибокого навчання. Тисячі веб-сайтів запускаються

щодня, а це означає, що навчання можна проводити за допомогою сучасних елементів дизайну.

2.2 Порівняння розробки на основі ШІ та веб-розробника

Ще 20 років тому створення сайтів було неймовірно складним процесом, який могли зробити лише одиниці людей. Потенційним замовникам доводилося витрачати купу грошей, щоб їхній сайт був втілений у життя будь-якою веб студією.

З приходом нових технологій, бібліотек, фреймворків, препроцесорів та іншого створення сайтів стало набагато простіше. Сьогодні вже не треба витрачати багато грошей і часу на створення сайтів, адже можна просто скористатися навіть простим конструктором на базі ШІ.

Якщо порівняти роботу веб-розробника і ШІ за декількома характеристиками, такими як витрати, дизайн та програмний код, а також за ефективністю використання часу, то маємо такі висновки:

− ефективність часу − безумовно, ШІ перевершує навіть найдосвідченіших фахівців у цій галузі. Це ще один інструмент, що наближає створення сайтів до повністю автоматизованого процесу;

− економічна ефективність − одна з найголовніших ознак, за допомогою систем на основі ШІ, можна створити і запустити свій веб-сайт за мінімальною ціною. В порівнянні з професійними командами веб-розробників це більш доступне рішення;

− проектування − ШІ насправді не розробляє ваш сайт, а вибирає дизайн зі зразків у базі даних. Тобто, веб-дизайнери повинні створювати нові шаблони і наповнювати базу даних, які будуть коректно застосовуватися до будь-якого контенту. Слід підкреслити, що веб-сайти,

розроблені на основі ШІ, обтяжені великою кількістю проблем, таких як неструктурований програмний код, який буде доволі складно в подальшому змінювати. Утім, для більшості користувачів такий варіант буде прийнятним;

− щодо дизайну і програмного коду, то тут перше місце, як і раніше, дістається розробникам. ШІ пише код без урахування семантичної розмітки, що важливо для структурування даних і посилення їх внутрішнього значення [9].

Отже, виходячи з вищезазначених висновків, немає сумнівів у тому, що сайти розроблені за допомогою ШІ, надзвичайно прості та корисні в розробці. Кількість підприємств, які бажають створити свої веб-сайти, зростає. Згідно з дослідженням, проведеним Clutch, в 2016 році 46 відсотків малих підприємств не мали веб-сайтів, але це число скоротилося до 29 відсотків всього за один рік [14]. Такий величезний попит змусить людей, які керують цим бізнесом, шукати дешеві і швидкі рішення.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розробка середовища для роботи нейронної мережі

Під час навчання нейронної мережі їй надається декілька зображень. Так система вчиться послідовно передбачати всі теги розмітки, що використовуються. При передбаченні наступних тегів алгоритму видається скріншот з «правильними» тегами розмітки.

На сьогоднішній день активно використовуються моделі, що передбачають черговість слів. Для кожного прогнозу алгоритму видається однаковий скріншот. Нейронна мережа створює ознаки даних. Ці ознаки формують зв'язки між вхідними та вихідними значеннями. Для визначення змісту малюнка мережа створює уявлення даних, виходячи з яких генерується HTML синтаксис передбачення. Так у алгоритму з'являються необхідні знання для передбачення наступного тега [3].

Використання вже тренованих мереж на вирішення практичних завдань мало відрізняється від навчання моделі з нуля. Текст генерується покроково, супроводжується однаковими картинками. Замість згодовування алгоритму правильних HTML тегів вона виходить вже згенеровану розмітку. Далі − передбачається наступний тег. Передбачення починається «початковим тегом» і закінчується «кінцевим тегом», або досягненням максимальної межі [7].

Для початку роботи з штучною нейронною системою створимо версію гіпертекстового документу з довільним змістом. Далі передамо документ до нейронної мережі і розпочнемо роботу зі створення рішення. Нейронна мережа відображає макет проекту в список значень пікселів. Від 0 до 255 px трьох каналах − синій, зелений і червоний.

Для уявлення розмітки в розумінні нейронної мережі, звернемося до технології палкого кодування (від англ. Hot Encode). Таким чином, бачимо відображення розмітки в пам’яті нейронної мережі (рисунок 3.1).

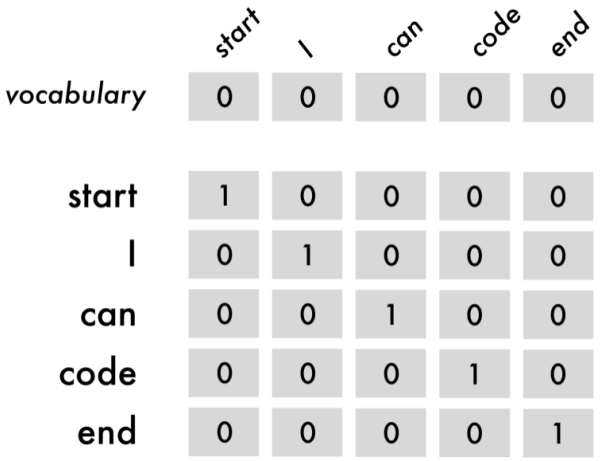

Рисунок 3.1 – Розмітка в пам’яті нейронної мережі

На рисунку вище доданий початковий та кінцевий теги. Дані теги – це деякі сигнали, що пояснюють алгоритму, коли потрібно розпочати і закінчити прогноз. Для вхідних значень візьмемо речення, що починаються з одного слова, та був послідовно додамо інші слова. Вихідне значення – це одне слово.

Пропозиції підпорядковуються тій самій логіці, як і слова. Їх також потрібна певна довжина вхідного значення. Замість обмежень щодо набору слів (словника) ми робимо прив'язку за максимальною довжиною речення. Якщо речення коротше максимальної довжини, воно доповнюється «порожніми» словами, тобто. словами з одними нулями (рисунок 3.2).

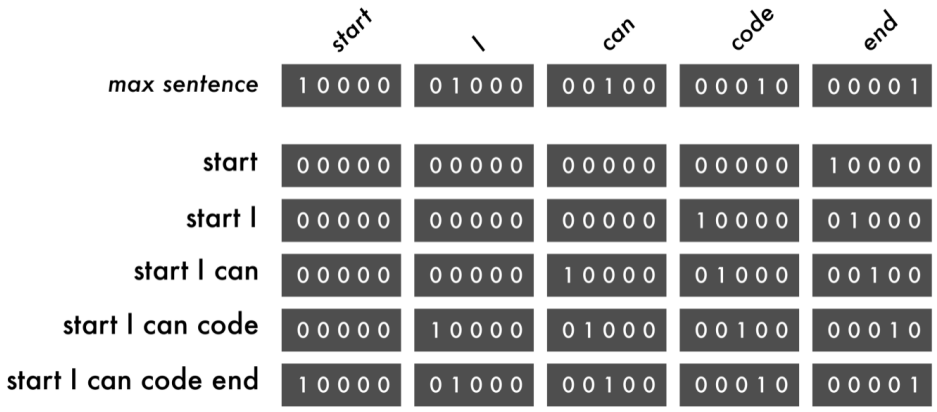

Рисунок 3.2 – Відображення моделі послідовності

Слова друкуються праворуч наліво. Таким чином, у кожному циклі навчання всі слова змінюють своє становище. Цей спосіб дозволяє алгоритму запам'ятовувати цілу послідовність, а чи не окреме розташування кожного слова.

На рисунку нижче наведено чотири прогнози. Кожен рядок – одне передбачення алгоритму. Ліворуч розташовані картинки у трьох колірних каналах: червоному, зеленому, синьому та попередні слова. За дужками розташовані послідовні прогнози; червоними квадратами позначено закінчення (рисунок 3.3). Програмний код приведений у лістингу 3.1.

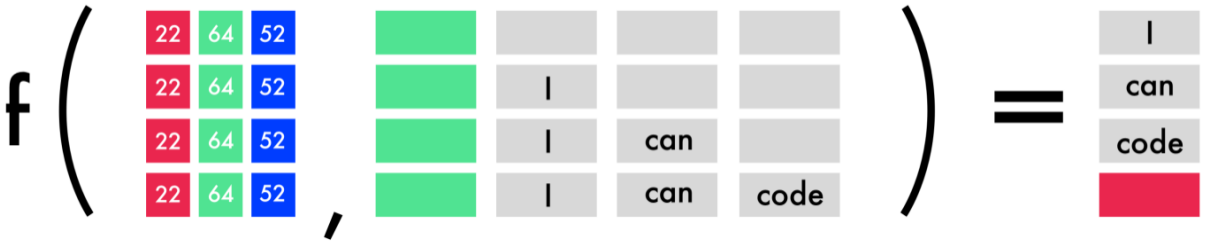

Рисунок 3.3 – Варіанти прогнозування рішення

Лістинг 3.1 – Програмний код алгоритму передбачення рішення

```
    # Вилучення ознак із зображення
    featuresList = VGG.predict(imagesSet)
    # Завантаження ознаки мережі, накладання Dense-шару та повторення вектора
    vggInput = Input(shape=(1000,))
    denseVgg = Dense(5)(vggInput)
    repeatVectorVgg = RepeatVector(max_caption_length)(denseVgg)
    # Вилучення інформації з вхідної послідовності
    textInput = Input(shape=(vocabSize, vocabSize))
    textModel = LSTM(5, return_sequences=True)(textInput)
    # Узагальнення інформації із зображення та вхідного значення та вилучення інформації з узагальненого виходу
    decoder = LSTM(5,return_sequences=False) (concatenate([repeatVectorVgg, textModel]))
    # Передбачення наступного слова
    decoderOutput = Dense(vocabSize, activation='softmax')(decoder)
    # Компіляція та запуск нейронної мережі
    model = Model(inputs=[vggInput, textInput], outputs = decoderOutput)
    model.compile(loss='categorical_crossentropy', optimizer = 'rmsprop')
```

3.2 Програмне забезпечення

3.2.1 Python

Під час створення інтелектуального парсеру для перетворення макетів web-сторінок у HTML розмітку була використана мова

програмування Python. Python – це інтерпретована, інтерактивна, об’єктно-орієнтована мова програмування загального призначення та високорівнева мова. Він був створений Гвідо ван Россумом у 1985-1990 роках. Вихідний код Python також доступний під загальною суспільною ліцензією GNU (GPL) [13]. Перелічимо деякі з ключових переваг вивчення Python:

– інтерпретованість – інтерпретатор обробляє вихідний файл під час виконання, він читає рядки коду один за одним і виконує те, що написано. Подібно до Perl та PHP, Python не вимагає компіляції програми перед її виконанням. Отже, не потрібно викликати компілятор. Замість запуску компілятора, який допомагає перетворювати вихідні файли в скомпільовані файли класів, тобто треба просто запустити файл .py. Компіляція байтового коду Python є автоматичною і повністю неявною;

– високорівневість – Python спирається на легкозчитувані структури, які пізніше перекладаються на мову низького рівня, оригінальний код, який запускається на центральному процесорному блоці комп’ютера. Мова високого рівня призначена для використання програмістом, а написаний код далі інтерпретується мовою низького рівня. Як і C++ або Java, перед запуском Python потрібно обробити. Це забезпечує портативність Python – вона може працювати на різних типах комп’ютерів майже без змін;

– загальне призначення – Python можна використовувати майже для всього. Він застосовний майже в кожній галузі для виконання різноманітних завдань. Будь то виконання таких короткострокових завдань, як тестування програмного забезпечення або довгострокова розробка продукту. Мова популярна не тільки серед інженерів-програмістів, а й серед фахівців інших галузей: математики, аналізу даних, науки, бухгалтерського обліку, мережевої інженерії;

– об’єктно-орієнтованість – ця парадигма програмування дає загальну орієнтацію на написання сценаріїв і потужне структурування коду. Цей об'єктно-орієнтований підхід дозволяє описувати проблеми в термінах класів і об'єктів. Потім об’єкти складаються таким чином, щоб писати складні комп’ютерні програми. Окрім об’єктно-орієнтованого програмування, Python також підтримує процедурну парадигму. Оскільки ООП є лише одним із варіантів, можна зробити програмування на Python більш просунутим, скориставшись підходом до об’єктно-орієнтованого програмування. Розробники можуть створювати багаторазові шаблони коду, таким чином зменшуючи надмірність у проектах розробки.

Python можна використовувати як мову сценаріїв або компілювати в байт-код для створення великих програм. Вона забезпечує дуже високорівневі динамічні типи даних і підтримує динамічну перевірку типів. Також мова підтримує автоматичний збір сміття. Її можна легко інтегрувати з C, C++, COM, ActiveX, CORBA та Java.

Python – це високорівнева, універсальна і дуже популярна мова програмування. Мова програмування Python (останній Python 3) використовується у веб-розробці, додатках машинного навчання, а також усі передові технології в індустрії програмного забезпечення. Мова програмування Python дуже добре підходить для початківців, а також для досвідчених програмістів з іншими мовами програмування, такими як C++ і Java.

Існує величезна різноманітність варіантів використання Python в різних галузях. Звичайно, перше, що спадає на думку про найпоширеніші способи використання Python, – це створення веб-додатків, мобільних і настільних додатків, а також їх тестування. Але Python – це мова, яка слугує багатьом цілям. В основному, використання Python ідеально

підходить для розробки веб-додатків, Data science, написання скриптів, програмування баз даних, швидкого створення прототипів.

Отже, Python добре підходить для всіх форм програмування, завдяки чому його база користувачів швидко зростає. Ось деякі з прикладів: кросплатформні сценарії оболонки, швидка автоматизація, проста веб-розробка, аналіз і візуалізація даних, AI та ML.

Фахівці використовують Python для кращого виконання різноманітних завдань у різних дисциплінах. Кращої продуктивності, серед іншого, можна досягти за допомогою автоматизації. Фінанси, страхування та маркетинг – це основні сфери, у яких люди стикаються з необхідністю виконувати повторювані й нудні завдання: переглядати, копіювати, перейменовувати та завантажувати файли на сервер, завантажувати веб-сайти чи аналізувати дані. Замість цього програміст може написати сценарій на Python і автоматизувати все це.

Не дарма найбільші компанії світу використовують Python. Pixar використовує його для створення фільмів, Google – для сканування сторінок, Netflix – для доставки вмісту, а Spotify – для рекомендації пісень. Ця мова сповнена переваг, і є кілька вагомих причин любити її:

– простота. Зрозумілий і простий синтаксис Python – це те, що змушує початківців вивчити цю мову сценаріїв. З певної точки зору, може здатися природним і наперед визначеним, що Python може перетворитися на «lingua franca» кодування, виявляючи, що всі його опоненти застаріли. Її код легко зрозуміти, поділитися та підтримувати. Тут немає багатослівності, а мову легко вивчити;

– потужний набір інструментів. За своєю суттю програми на Python є текстовими файлами, що містять інструкції для інтерпретатора, записані в текстовому редакторі або IDE. IDE є повнофункціональними і пропонують

вбудовані інструменти, такі як перевірка синтаксису, налагоджувачі та браузери коду, текстові редактори зазвичай не включають функції IDE, але їх можна налаштувати. Python також має величезний набір сторонніх пакетів, бібліотек і фреймворків, які полегшують процес розробки. Таким чином, ці можливості оптимізації роблять Python чудовим для великомасштабних проектів;

– швидкість розвитку. Python – це динамічна скриптова мова, тому вона не призначена для написання програм з нуля, але в першу чергу призначена для з'єднання компонентів. Компоненти призначені для повторного використання, а інтерфейси між компонентами та сценаріями чітко визначені. Все це прискорює швидкість розробки програмного забезпечення за допомогою Python, роблячи мову дуже стислою та продуктивною;

– гнучкість. Хоча Python робить акцент на простоті та читабельності коду, а не на гнучкості, мова все ще має це. Python можна використовувати в різних проектах. Це дозволяє розробникам вибирати між об'єктно-орієнтованим і процедурним режимами програмування. Python також гнучкий у типі даних. Їх п'ять: число, рядок, список, кортеж і словник, і кожен підтип даних відповідає одному з цих кореневих типів. В результаті дослідницький аналіз даних стає легшим у виконанні завдяки гнучкості Python;

– портативність. Python розроблено для переносимості. Її програми підтримуються будь-якою сучасною ОС комп'ютера. Завдяки високорівневій природі мови, сценарій Python інтерпретується, тому його можна написати для подальшої інтерпретації однаково добре в Linux, Windows, Mac OS і UNIX, не вимагаючи змін. Програми Python також дозволяють реалізувати портативні графічні інтерфейси;

– сильна громада. Python має швидко зростаючу базу користувачів і насправді є репрезентацією того, що таке сильна спільнота. Існують тисячі співробітників потужного інструментарію Python – Pythonists. Вже існує майже 200 000 спеціально створених програмних пакетів, завантажених користувачами в онлайн-сховище. Усе це означає, що велика спільнота підтримки є як причиною, так і наслідком того, що мова затребувана [16].

На додачу, мова Python містить NumPy, що був використаний під час створення інтелектуального парсеру для перетворення макетів web-сторінок у HTML розмітку. NumPy – це бібліотека Python, яка використовується для роботи з масивами. Вона також має функції для роботи в області лінійної алгебри, перетворення Фур'є та матриць. NumPy частково написана на Python, але більшість частин, які вимагають швидкого обчислення, написані на C або C++.

NumPy було створено у 2005 році Тревісом Оліфантом. Це проект з відкритим кодом, і ви можете використовувати його вільно. NumPy розшифровується як числовий Python. У Python є списки, які служать меті масивів, але вони повільно обробляються, в той час як NumPy має на меті надати об’єкт масиву до 50 разів швидше, ніж традиційні списки Python.

Об’єкт масиву в NumPy називається ndarray, він надає безліч допоміжних функцій, які роблять роботу з ndarray дуже легкою. Масиви дуже часто використовуються в науці про дані, де швидкість і ресурси дуже важливі.

На відміну від списків, масиви NumPy зберігаються в одному безперервному місці в пам’яті, тому процеси можуть дуже ефективно отримувати доступ до них і маніпулювати ними. Така поведінка в інформатиці називається локальністю посилання. Це основна причина,

чому NumPy швидше, ніж списки. Також ця бібліотека оптимізована для роботи з останніми архітектурами ЦП.

3.2.2 Jupyter Notebook

Jupyter Notebook – програма з відкритим вихідним кодом, в якій можна відразу побачити результат виконання коду (у цьому плані це незвичне для всіх IDE).

Головна відмінність від традиційних інструментів розробки – можливість розбити код на частини та виконувати їх окремо. Наприклад, ви можете написати одну функцію і одразу перевірити, як вона працює, не запускаючи решту фрагментів коду. Можна змінювати порядок виконання коду. І, найголовніше, – увесь код можна змінити динамічно. Тобто, після переписування фрагменту коду, його можна знову запустити та побачити інший результат.

На додачу, відображення виводу результатів відбувається відразу під виконаним фрагментом, завдяки цьому Jupyter ноутбук дуже популярний в аналітиці даних та Data Science. Фахівці отримують попередні результати, будують графіки та іншу візуалізацію.

Після повторного відкриття файлу, він не лише покаже код, що був написаний у всіх різних осередках, але й результат, що залишився з моменту останнього виконання. Це дуже корисно, коли є необхідність програмувати не для створення програм, а для виконання та аналізу математичних та/або технічних операцій.

Основна мова Jupyter Notebook – Python. Це пов'язано з тим, що він є наступником більш старого проекту IPython Notebook. Але інші мови також підтримуються.

Можливо, машинне навчання та Data Science – найбільші області застосування Jupyter. Але він може бути неймовірно корисним майже в кожній програмі на Python, де ціль полягає в тому, щоб запустити програму і побачити результат без створення кінцевого продукту.

Незважаючи на всі переваги, у розробників ставлення до Jupyter Notebook неоднозначне. Наприклад, його не рекомендують використовувати як середовище розробки. Причин кілька:

– у Jupyter Notebook пишеться не програма, а окремі шматки коду, які запускаєте ізольовано один від одного. Через це виходить дуже складний глобальний стан. Код у осередках редагується, змінюється. Коли осередків стає занадто багато, буває складно зрозуміти, звідки беруться змінні і як було досягнено результату.

– з першої проблеми випливає інша – погана переносимість. Код Jupyter Notebook не можна просто скопіювати у файл і відкрити в іншому середовищі розробки. Його доведеться частково переписувати.

Основні галузі застосування – нейромережі, машинне навчання, аналіз та візуалізація даних, робота зі статистикою. Це підтверджує дослідження репозиторіїв GitHub. Наприклад, у 2020 році найбільше зірок на Github отримав проект Fastbook. Це введення у глибоке навчання за допомогою бібліотек Pytorch та FastAI, опубліковане за допомогою Jupyter Notebook.

Найпростіший спосіб запустити Jupyter Notebook – використовувати онлайн-сервіси. Найбільш поширене рішення – хмара Google Colaboratory або скорочено Colab. Це безкоштовний сервіс, для роботи з яким потрібний лише обліковий запис Google.

3.2.3 TensorFlow

Кілька років тому глибоке навчання почало перевершувати всі інші алгоритми машинного навчання, надаючи безліч даних. Google побачив, що може використовувати ці глибокі нейронні мережі для покращення своїх послуг. Він створив середовище під назвою Tensorflow, щоб дозволити дослідникам та розробникам спільно працювати над моделлю ШІ. Після розробки та масштабування це дозволо багатьом людям використати дане середовище.

Вперше його опублікували наприкінці 2015 року, а перша стабільна версія з'явилася в 2017 році. Це вихідний вихідний код під ліцензією Apache Open Source. Ви можете використовувати його, змінювати його та розповсюджувати змінену версію, нічого не платячи Google.

Архітектура Tensorflow працює у трьох частинах:

– попередня обробка даних;

– побудова моделі;

– навчання та тестування моделі.

Програмна бібліотека називається Tensorflow, тому що вона приймає вхідні дані у вигляді багатовимірного масиву, також відомого як тензори. Можна створити свого роду блок-схему операцій. Ввід відбувається на одному кінці, а потім проходить через цю систему безлічі операцій і виходить на іншому кінці як висновок.

Вимоги до обладнання та програмного забезпечення TensorFlow можна розділити на:

– етап розробки: режим тренування. Навчання зазвичай проводиться на робочому столі або ноутбуці;

– фаза запуску або фаза виведення: після закінчення навчання Tensorflow можна запускати на різних платформах, таких як робочий стіл під керуванням Windows, MacOS або Linux, хмарі як веб-сервісу або мобільних пристроях, таких як iOS та Android.

Середовище можна навчити на декількох машинах, та запустити на іншій машині, як тільки буде навчена модель.

Модель можна навчати та використовувати як на GPU, так і на CPU. Графічні процесори були призначені для відеоігор. Наприкінці 2010 року дослідники зі Стенфорда виявили, що GPU також дуже гарний у матричних операціях та алгебрі, що робить їх дуже швидкими для виконання подібних обчислень. Глибоке навчання ґрунтується на множенні матриць. TensorFlow дуже швидко обчислює множення матриць, тому що воно написане на C++. Хоча реалізований на C ++, TensorFlow може бути доступний і керуватися в основному іншими мовами Python.

TensorFlow є однією з найкращих відкритих бібліотек для чисельних обчислень. Такі гіганти, як DeepMind, Uber, AirBnB або Dropbox, вибрали цей фреймворк для своїх потреб.

Бібліотека Tensorflow включає різні API для побудови в масштабі архітектури глибокого навчання, таких як CNN або RNN. TensorFlow базується на обчисленні графа; це дозволяє розробнику візуалізувати будову нейронної мережі за допомогою Tensorboad. Цей інструмент корисний для налагодження програми. TensorBoard дозволяє графічно та візуально відстежувати, що робить TensorFlow. Нарешті, Tensorflow створено для розгортання масштабу. Він працює на процесорі та графічному процесорі.

Tensorflow приваблює найбільшу популярність на GitHub, порівняно з іншими системами глибокого навчання.

В даний час TensorFlow 1.10 має вбудований API для:

– лінійної регресії: tf.estimator.LinearRegressor;

– класифікації: tf.estimator.LinearClassifier;

– класифікації глибокого навчання: tf.estimator.DNNClassifier;

– tf.estimator.DNNLinearCombinedClassifier, що поєднує DNN та лінійну класифікацію;

– прискорення регресії дерева: tf.estimator.BoostedTreesRegressor

– посилена класифікація дерева: tf.estimator.BoostedTreesClassifier.

Бібліотека TensorFlow має безліч плюсів:

– для неї написано велику кількість посібників та документації;

– вона пропонує потужні засоби моніторингу процесу навчання моделей та візуалізації (Tensorboard);

– підтримується великою спільнотою розробників та технічними компаніями;

– забезпечує обслуговування моделей;

– підтримує розподілене навчання;

– TensorFlow Lite забезпечує виведення на пристрої із низькою затримкою для мобільних пристроїв;

Також наявні такі мінуси, як:

– програє за швидкістю роботи у еталонних тестах, проти CNTK і MXNet, наприклад;

– має вищий вхідний поріг для початківців, ніж PyTorch або Keras. Гола Tensorflow є досить низькорівневою і вимагає багато шаблонного коду, і режим «визначити і запустити» для Tensorflow значно ускладнює процес дебагу.

Є ще одне значне обмеження: єдина повністю підтримувана мова – Python.

3.2.4 Keras

Keras – це бібліотека для Python, що дозволяє легко та швидко створювати нейронні мережі. Вона сумісна з TensorFlow, Theano, Microsoft Cognitive Toolkit та MXNet. Перші дві платформи найбільш використовуються розробки алгоритмів глибокого навчання, але досить складні в освоєнні. Keras ж, навпаки, є чудовим варіантом для тих, хто тільки починає вивчення нейронних мереж.

Автор бібліотеки Франсуа Шолле, інженер Google, розробив Keras для того, щоб максимально спростити процес створення нейронних мереж. Наголос був на розширюваності, модульності та мінімалізмі з підтримкою Python.

Розробка Keras дозволила компанії Google зробити великий внесок у глибоке навчання та штучний інтелект. Пов'язано це насамперед із тим, що бібліотека містить сучасні алгоритми, які раніше були недоступні.

З особливостей прийнято виділяти такі фактори:

– зручність користувача. Keras був розроблений для користувачів, а не для машин – він використовує особливі методи: пропонує узгоджений та простий API, мінімізує кількість дій користувача, необхідних для вирішення найпоширеніших завдань. У разі виникнення помилок завжди можна звернутися на підтримку зворотного зв'язку;

– модульність. Під цим розуміється послідовність або граф автономних, повністю налаштованих модулів, які можуть бути підключені без додаткових обмежень. Наприклад, це можуть бути нейронні шари, функції помилки, оптимізатори, схеми ініціалізації, функції активації та схеми регулювання – всі ці модулі можна комбінувати для створення моделі;

– розширюваність. Можливість легко додавати нові класи, модулі та функції робить Keras відмінним засобом для проведення різноманітних досліджень;

– робота з Python. Всі моделі були написані мовою програмування Python, тому код завжди компактний та легко читається.

В основі Keras лежать моделі, основний тип яких – послідовність, що є лінійним стек шарів.

Суть полягає в наступному: створюється послідовність і до неї додаються шари тому порядку, у якому потрібно виконати обчислення. Після визначення компілюється модель, що використовує базову платформу оптимізації обчислень.

Далі модель повинна відповідати даним – це можна зробити по одній партії даних за один раз або запустивши весь режим навчання моделі. Коли навчання буде завершено, можна використовувати модель для прогнозування нових даних.

Також є ще один тип моделей – це клас Model, який використовується з функціональним API.

Цей фреймворк гарний у кейсах для перекладу, розпізнавання зображень, мовлення тощо.

Keras має наступні переваги:

– прототипування дійсно швидке та просте;

– досить незначний для побудови моделей глибокого навчання для безлічі шарів;

– має модулі, що повністю конфігуруються;

– має простий та інтуїтивно-зрозумілий інтерфейс, відповідно, добрий для новачків;

– має вбудовану підтримку для навчання на кількох GPU;

– може бути налаштований як оцінювач для TensorFlow і навчений на кластерах GPU на платформі Google Cloud;

– запускається на Spark;

– підтримує GPU від NVIDIA, TPU від Google, GPU із Open-CL, такі як AMD.

Із недоліків маємо:

– може бути занадто високорівневим і не завжди легко кастомізується;

– обмежений бекендами Tensorflow, CNTK та Theano.

Keras не настільки функціональний як TensorFlow і дає менше опцій для керування мережевим з'єднанням, що може стати серйозним обмеженням, якщо існує намір створити якусь спеціалізовану модель глибокого навчання.

Отже, розглянуті основні інструменти для створення програмної системи, зокрема мова програмування Python, бібліотека машинного навчання TensorFlow та відкрита нейромережна бібліотека Keras. В результаті сформований вибір інструментів для створення програмної системи.

3.3 Формат даних

На вхід алгоритму створеного проекту подається набір пар «HTML код» – «зображення». Після чого за допомогою енкодера проводиться аналіз зображень для виведення на них ознак. Після цього процесу отримані ознаки набувають зв’язок з тегами HTML-коду. Це є проміжними даними для моделі.

Вихідними даними алгоритму є HTML-код, що отримано на основі аналізу зображення, а саме пошуку ознак на ньому, послідовному їх аналізі і прогнозуванню наступного тега.

3.4 Побудова архітектури проекту

Розглянемо загальний алгоритм роботи проекту, що складається з наступних етапів:

а) підключення бібліотек;

б) визначення функцій;

в) підготовка даних;

1) завантаження зображень і текстів;
2) виділення властивостей на зображеннях;
3) токенізація текстів;
4) ініціалізація даних у відповідному для навчання вигляді;

г) створення структури нейронної мережі з шарів Keras;

ґ) тренування нейронної мережі зі збереженням результатів;

д) тестування.

Було використано наступні основні бібліотеки:

– numpy;

– keras;

– tensorflow.

Ці бібліотеки дозволяють зручно працювати з масивами даних, попередньо обробляти їх, створювати алгоритми машинного навчання, складаючи з компонентів, використовувати готові алгоритми.

Серед визначених функцій є прості функції для завантаження файлу та кодування слів у числові ідентифікатори. Також є функція генерації

HTML-коду generate_text(), що використовує натреновану нейронну мережу, токенізовані теги і фото для генерації з нього самого коду. У функції послідовно, тег за тегом, генерується HTML-код, орієнтуючись на результати тренування нейронної мережі.

Після завантаження всіх файлів: зображень і текстів HTML проводиться їхня попередня обробка. Для визначення властивостей зображень використано алгоритм InceptionResNetV2. Цей алгоритм повертає модель класифікації зображень Keras, за бажанням завантажений вагами, попередньо натренованими в ImageNet.

Inception-ResNet-v2 є згортковою нейронною мережею, яка навчена більш ніж на мільйон зображень від бази даних ImageNet. Мережа складається із 164 шарів глибиною і може класифікувати зображення в 1000 категорій об'єктів. В результаті мережа навчилася багатим представленням функцій для широкого діапазону зображень.

Також всі тексти HTML токенізуються для створення ідентифікаторів всіх слів (тегів) у них.

Після всіх маніпуляцій з даними вони записуються до наборів тренувальних даних, що будуть передані до нейронної мережі.

Існують два основні розділи спроектованої нейронної мережі, які відповідають за побудову HTML документа. Перший – енкодер. Там ми створюємо ознаки зображення та попередньої розмітки. Ознаки – це основні елементи, які створюються мережею для визначення зв'язків між макетом і розміткою. Наприкінці кодування ми прив'язуємо ознаки зображення до кожного слова попередньої розмітки.

Потім декодер бере об'єднані ознаки макета та розмітки та створює ознаку наступного тега. Ця функція запускається через повністю

підключену нейронну мережу та виконує передбачення наступного тега (рисунок 3.4).

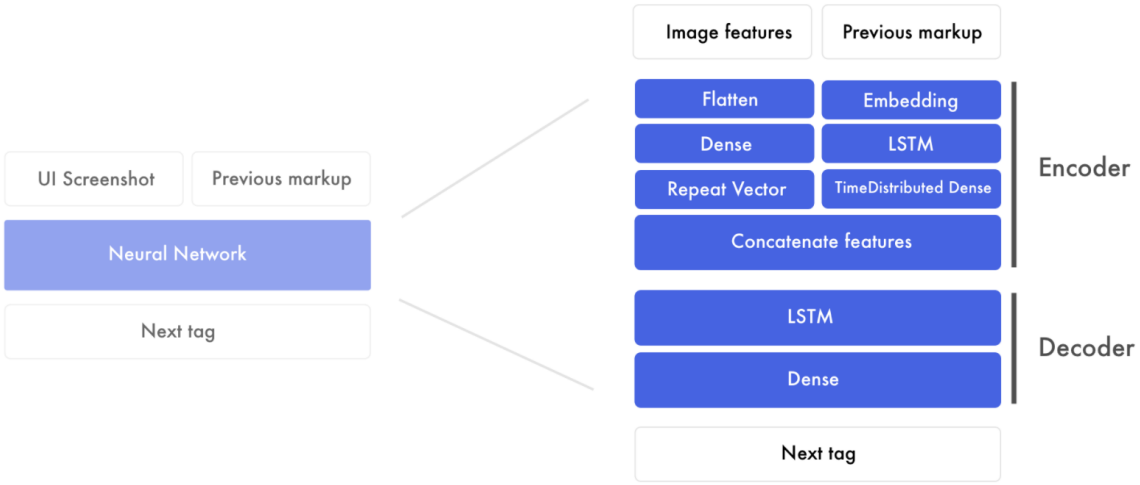

Рисунок 3.4 – Основні розділи побудови HTML документа

У попередньому прикладі уявлення розмітки було виконано за допомогою прямого кодування (метод one-hot). Тепер слова додані у вхідні значення, але на виході використовується пряме кодуванням.

Таким чином, ми задаємо незмінність пропозиції, але різне уявлення токенів. Пряме кодування сприймає кожне слово як незалежну одиницю. Тому кожне слово перетвориться із вхідних значень до числового списку. Цей список створює зв'язок між тегами розмітки.

Keras – бібліотека нейронної мережі високого рівня, Keras написана на чистому Python і заснована на Tensorflow або Theano. Основна структура даних Keras – це «модель», яка є способом організації мережевого рівня. Основною моделлю в Keras є послідовна модель, що складається з набору послідовних мережевих рівнів.

Основна концепція Keras – бути короткою і простою у використанні, забезпечуючи при цьому абсолютний контроль користувачів над Keras.

Користувачі можуть налаштовувати власні моделі, мережеві рівні і навіть змінювати вихідний код відповідно до своїх потреб.

Шари є основними елементами, необхідними при створенні нейронних мереж. Наукові шари відповідають за архітектуру моделі глибокого навчання. Кожен із них виконує вирахування на основі даних, отриманих з попереднього. Потім інформація передається далі. Нарешті, останній шар видає потрібний результат.

Для визначення або створення шару Keras потрібна наступна інформація:

– форма вводу: для розуміння структури вхідної інформації;

– кількість: для визначення кількості вузлів/нейронів у шарі;

– ініціалізатор: для визначення вагів кожного входу, що важливо для виконання обчислень;

– активатори: для перетворення вхідних даних у нелінійний формат, щоб кожен нейрон був спроможний ефективно навчатися;

– обмежувачі: для накладень обмежень на ваги при оптимізації;

– регулятори: для застосування штрафів до параметрів у час оптимізації.

Рекурентні нейронні мережі (recurrent neural networks, RNN) – це мережі типу FFNN, але з особливістю: нейрони отримують інформацію не тільки від попереднього шару, а й від себе попереднього проходу. Це означає, що порядок, в якому подаються дані та навчається мережа, стає важливим [10]. Великою складністю мереж RNN є проблема зникаючого (або вибухового) градієнта, що полягає у швидкій втраті інформації з часом. Звичайно, це впливає лише на ваги, а не стану нейронів, але саме в них накопичується інформація. Зазвичай мережі такого типу використовуються для автоматичного доповнення інформації [8].

Мережі з довгою короткостроковою пам'яттю (LSTM – Long Short Term Memory) вирішують проблему втрати інформації, що була описана вище, використовуючи явно задану клітину пам'яті та фільтрацію.

У кожного нейрона є клітина пам'яті і три фільтри: вхідний, вихідний і забуваючий. Призначенням даних фільтрів є захист інформації. Вхідний фільтр вказує, яка інформація з попереднього шару зберігатиметься в клітині. Вихідний фільтр вказує, яку інформацію отримають такі шари. Забуваючий фільтр також виконує корисну функцію: наприклад, якщо мережа вивчає книгу і переходить на новий розділ, якісь символи зі старої можна забути. Такі мережі здатні навчитися створювати складні структури, а ресурсів вони споживають чимало.

Якщо вектор уявлення слова прогнати через довгу короткострокову пам'ять (LSTM), на виході отримаємо послідовність ознак розмітки. Все проходить через Dense-шар TimeDistributed – простіше кажучи, через Dense-шар із безліччю входів та виходів.

Dense-шар отримує інформацію з усіх вузлів попереднього шару. Паралельно відбувається зведення ознак зображення. Незалежно від кількості структурованих чисел, всі вони перетворюються на один довгий числовий список. Потім до шару застосовується Dense-шар та створюються ознаки високого рівня. Далі ці ознаки поєднуються в ознаки розмітки [17].

Тепер для передбачення наступного тега будемо користуватися об'єднаними ознаками розмітки. LSTM-шар має послідовність у значенні false. Замість повернення довжини послідовності на вході він передбачає ознаку. Dense-шар виступає у ролі традиційної нейронної мережі прямого поширення. Активація softmax в Dense-шарі розподіляє ймовірність від 0 до 1 із сумою всіх прогнозів, що дорівнює 1.

Шар Flatten використовується для «згладжування» введення, тобто для того, щоб зробити багатовимірне введення одновимірним, що часто використовується при переході від шару згортки до повністю пов'язаного шару. Flatten не впливає на розмір партії. Іншими словами, шар використовується для конвертації вхідних даних у меншій розмірі. Наприклад, вхідний шар розмірності (batch_size, 3,2) «вирівнюється» для виведення розмірів (batch_size, 6).

На рисунку зображено базові моделі: згорткова (CNN) та рекурентна (RNN) нейронна мережа. Остання найчастіше використовується в довгій короткочасній пам'яті (LSTM) (рисунок 3.5).

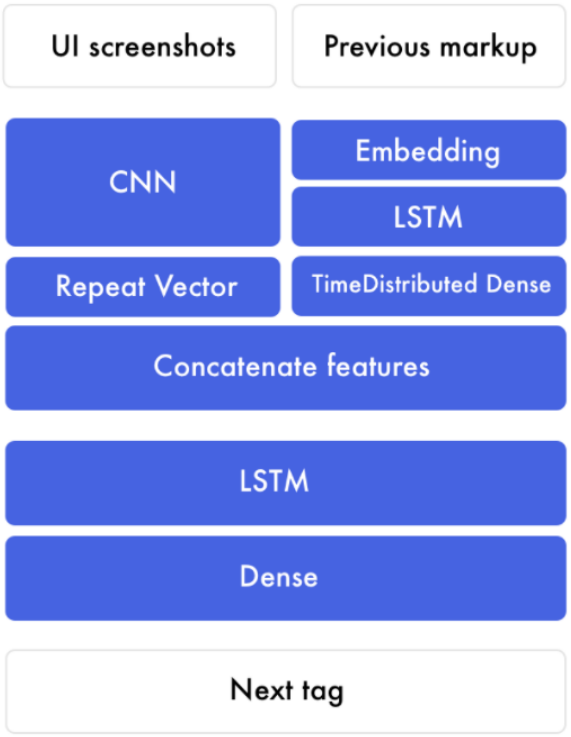

Рисунок 3.5 – Основні моделі нейронних мереж застосовуваних при створенні розмітки

Найважчим розділом LSTM є тимчасові кроки. LSTM складається з вхідних значень з тимчасовими кроками. По суті, ця та сама нейронна мережа, але налаштована на показ інформації у певному порядку.

Якщо розгорнути дану модель, то для кожного спадаючого кроку зберігатиметься однакове значення. Для попереднього вихідного результату та нового вхідного значення беруться різні набори вагових параметрів (рисунок 3.6).

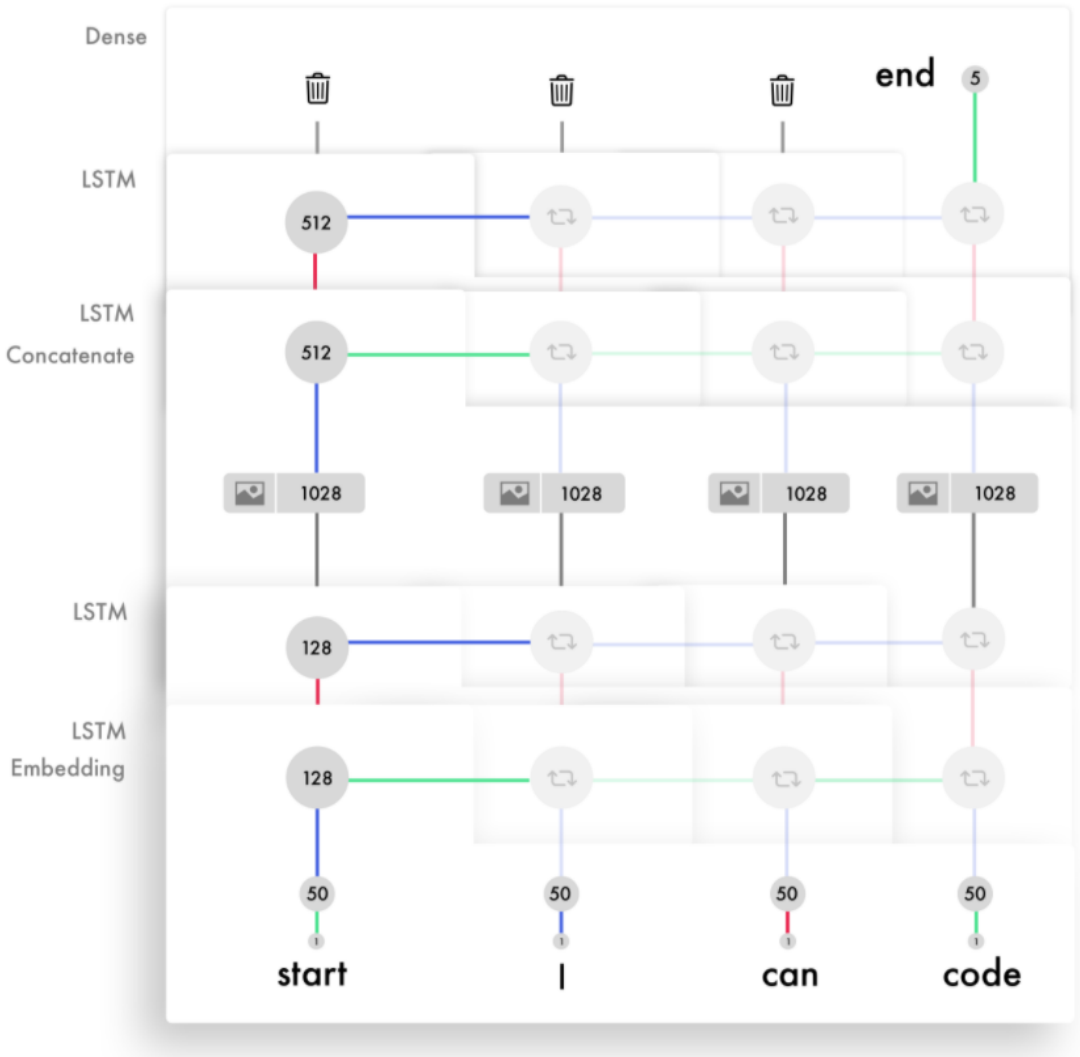

Рисунок 3.6 – Версія процесу для кожного тимчасового кроку в LSTM

Вхідні та вихідні вагові значення об'єднуються та додаються за допомогою активації. Це вихідне значення для цього тимчасового кроку. А оскільки вагові коефіцієнти використовуються кілька разів, то вони беруть інформацію з різних вхідних значень і створюють своє знання про послідовність.

Кількість рівнів у кожному шарі LSTM визначає її здатність до запам'ятовування. Воно також залежить від розміру кожної вихідної ознаки. Ознака – це довгий список цифр, які використовуються для передачі між шарами.

Кожен рівень LSTM підтримує стан осередку (тобто пам'ять). Для зміни даного стану використовуються активації та вагові значення. Це дозволяє LSTM-шарам «фільтрувати» інформацію для кожного вхідного значення на потрібну та непотрібну.

Крім проходження через вихідні ознаки кожному входу ведеться передача станів осередку – за одним значенням кожного рівня в LSTM [18].

По-перше, спочатку створюється енкодер для аналізу зображення і пошуку на ньому ознак. У ньому використано згорткові шари для аналізу зображення, вирівнюючі для перетворення вхідних даних у одновимірні та повнозв’язні шари.

Наступним кроком, для створення декодера проектується мережа з використанням LSTM шарів, що здатні на аналіз послідовних даних. Ця модель поєднується з іншою енкодером і має у собі ще декілька LSTM шарів після поєднання мереж.

У представленому коді створення моделі нейронної мережі описано всі частини моделі (рисунок 3.7).

```
#Create the encoder
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='valid', activation='relu', input_shape=(8, 8, 1536,)))
model.add(Conv2D(32, (3,3), activation='relu', padding='same', strides=2))
model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
model.add(Conv2D(64, (3,3), activation='relu', padding='same', strides=2))
model.add(Conv2D(128, (3,3), activation='relu', padding='same'))
model.add(Conv2D(128, (3,3), activation='relu', padding='same', strides=2))
model.add(Conv2D(128, (3,3), activation='relu', padding='same'))

model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.3))

model.add(RepeatVector(max_length))
input = Input(shape=(8, 8, 1536,))
encoder = model(input)

text_input = Input(shape=(max_length,))
text_model = Embedding(vocab_size, 50, input_length=max_length, mask_zero=True)(text_input)
text_model = LSTM(256, return_sequences=True)(text_model)
text_model = LSTM(256, return_sequences=True)(text_model)

#Create the decoder
decoder = concatenate([encoder, text_model])
decoder = LSTM(1024, return_sequences=True)(decoder)
decoder = LSTM(1024, return_sequences=False)(decoder)
decoder = Dense(vocab_size, activation='softmax')(decoder)

# Compile the model
model = Model(inputs=[input, text_input], outputs=decoder)
optimizer = RMSprop(lr=0.0001, clipvalue=1.0)
model.compile(loss='categorical_crossentropy', optimizer=optimizer)
```

Рисунок 3.7 – Програмний код алгоритму передбачення з використанням LSTM

Для оптимізації нейронної мережі використовується алгоритм RMSprop. Це оптимізатор, який реалізує алгоритм RMSprop. Суть RMSprop полягає в тому, щоб:

– підтримувати змінюване середнє значення квадрата градієнтів;

– ділити градієнт на корінь із цього середнього значення.

Розглянемо аргументи даного алгоритму:

– Learning_rate: тензор, значення з плаваючою комою або розклад, який є tf.keras.optimizers.schedules. LearningRateSchedule, або виклик, який не приймає аргументів і повертає фактичне значення для використання. Швидкість навчання, що за замовчуванням 0,001;

– rho: коефіцієнт дисконтування для історичного/майбутнього градієнта, за замовчуванням 0,9;

– momentum: скаляр або скалярний тензор, за замовчуванням 0.0;

– epsilon: невелика константа для чисельної стабільності. За замовчуван- ням 1e-7;

– centered: булеве значення. Якщо True, градієнти нормалізуються за оціненою дисперсією градієнта; якщо False, до нецентрованого другого моменту. Встановлення значення True може допомогти під час навчання, але трохи дорожче з точки зору обчислень і пам’яті. За замовчуванням значення False;

– name: необов'язковий префікс імені для операцій, створених під час застосування градієнтів. За замовчуванням встановлено «RMSprop»;

– **kwargs: аргументи ключових слів. Дозволені аргументи: clipvalue, clipnorm, global_clipnorm. Якщо встановлено значення clipvalu (float), градієнт кожного ваги обрізається так, щоб він не перевищував це значення. Якщо встановлено clipnorm (float), градієнт кожного ваги обрізається окремо, щоб його норма була не вище цього значення. Якщо встановлено global_clipnorm (float), градієнт усіх ваг обрізається так, щоб їх глобальна норма була не вище цього значення.

В щільній (dense) реалізації цього алгоритму змінні та відповідні їм параметри (імпульс, градієнт ковзного середнього, квадратний градієнт ковзного середнього) будуть оновлюватися, навіть якщо градієнт дорівнює нулю (тобто параметри будуть спадати, буде застосовуватися імпульс).

Розріджена (sparse) реалізація (використовується, коли градієнт є об’єктом IndexedSlices, як правило, через tf.gather або вбудовування пошуку в прямому проході) не оновлюватиме змінні фрагменти або їх накопичувачі, якщо ці зрізи не були використані в прямому проході (також немає «можливого» виправлення для врахування цих пропущених оновлень). Це призводить до більш ефективних оновлень для великих таблиць пошуку вбудовування (де більшість зрізів не доступні під час виконання конкретного графіка), але відрізняється від опублікованого алгоритму (рисунок 3.8).

```
# Train the neural network
model.fit([image_data, X], y, batch_size=64, shuffle=False, epochs=500, callbacks=callbacks_list)
```

```
Epoch 1/500
11/11 [==============================] - 243s 22s/step - loss: 4.3164
Epoch 2/500
11/11 [==============================] - 235s 21s/step - loss: 4.1486
Epoch 3/500
11/11 [==============================] - 239s 22s/step - loss: 4.1183
Epoch 4/500
11/11 [==============================] - 238s 22s/step - loss: 4.0993
Epoch 5/500
```

Рисунок 3.8 – Процес тренування нейронної мережі

Після успішного тренування проводиться аналіз алгоритму, для чого використовується функція генерації HTML-коду з тестового зображення (рисунки 3.9, 3.10, 3.11).

Header

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu ante eget dolor fermentum tincidunt. Proin elementum mi arcu, in commodo urna tristique non. Proin vel pharetra metus, a mollis lacus. Vivamus sed lobortis elit. Donec elementum feugiat placerat. Curabitur pharetra elit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu ante eget dolor fermentum tincidunt. Proin elementum mi arcu, in commodo urna tristique non. Proin vel pharetra metus, a mollis lacus. Vivamus sed lobortis elit. Donec elementum feugiat placerat. Curabitur pharetra elit.

Column 1	Column 2	Column 3	Column 4
Value 1	Value 2	Value 3	Value 4

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu ante eget dolor fermentum tincidunt. Proin elementum mi arcu, in commodo urna tristique non. Proin vel pharetra metus, a mollis lacus. Vivamus sed lobortis elit. Donec elementum feugiat placerat. Curabitur pharetra elit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu ante eget dolor fermentum tincidunt. Proin elementum mi arcu, in commodo urna tristique non. Proin vel pharetra metus, a mollis lacus. Vivamus sed lobortis elit. Donec elementum feugiat placerat. Curabitur pharetra elit.

Footer

Рисунок 3.9 – Приклад зображення для генерації коду

```
# Get the image, preprocess it, extract features and convert them to HTML-code
test_image = preprocess_input(np.array(img_to_array(load_img(drive_path+images_path+'sample_1.png', target_size=(299, 299))), dtype=float))
test_features = IR2.predict(np.array([test_image]))
text = convertImgToHTML(model, tokenizer, np.array(test_features), max_caption_len)
print(text)
```

```
'<html> \n<head> \n    <style> table, th, td {border: 1px solid black} \n    </style>\n</head> \n<body> \n    <h1> Header </h1> \n    <div> \n
p> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu ante eget dolor fermentum tincidunt.\n                Proin elementum mi ar
vel pharetra metus, a mollis lacus.\n                Vivamus sed lobortis elit. Donec elementum feugiat placerat. Curabitur pharetra elit. </p>
0%> \n            <p> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu ante eget dolor fermentum tincidunt.\n                Pr
ristique non. Proin vel pharetra metus, a mollis lacus.\n                Vivamus sed lobortis elit. Donec elementum feugiat placerat. Curabitur
</div> \n    <table> \n        <tr> \n            <th> Column </th> \n            ...'
```

Рисунок 3.10 – Результат роботи алгоритму у вигляді HTML-коду

```
display(HTML(text))
```

Header

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu ante eget dolor fermentum tincidunt. Proin elementum mi arcu, placerat. Curabitur pharetra elit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu ante eget dolor fermentum tincidunt. Proin elementum mi arcu, placerat. Curabitur pharetra elit.

Column	Column	Column	Column
Value	Value	Value	Value

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu ante eget dolor fermentum tincidunt. Proin elementum mi arcu, placerat. Curabitur pharetra elit.

shutterstock.com · 323592404

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu ante eget dolor fermentum tincidunt. Proin elementum mi arcu, placerat. Curabitur pharetra elit.

Footer

Рисунок 3.11 — Результат роботи алгоритму у вигляді візуалізованого HTML-коду

Отже, після навчання нейронна мережа стала здатною аналізувати та розуміти вхідні дані, а потім відповідним чином повертати вихідні дані у вигляді перетворення макетів web-сторінок у HTML розмітку.

ВИСНОВКИ

В кваліфікаційній роботі було проаналізовано теоретичний матеріал, технічну літературу щодо обраної предметної галузі. Були сформовані вимоги та досліджено методи автоматичної розробки веб-сайтів. Відбулося знайомство з рішеннями, існуючими на даний час та знайдена наукова новизна задачі. На основі отриманих результатів розроблено модель інтелектуального парсеру для перетворення макетів web-сторінок у HTML розмітку, що за допомогою алгоритму дозволяє моделі достатньо точно передбачити веб-компоненти.

Рушійною силою великих відкриттів і досягнень в історії людства впродовж віків були технології. Сьогодні ми живемо в одну з найбільш захоплюючих епох, коли технологічний прогрес постійно робить величезні кроки вперед. Саме розвиток Штучного Інтелекту є однією з найбільш перспективних технологічних розробок.

Щоб створити нейронну мережу, яка може генерувати розмітку HTML/CSS за певним скриншотом, при навчанні нейронної мережі треба надати їй декілька зображень з правильно структурованим HTML. При передбаченні наступних тегів, алгоритм звертається до вже передбачених і складає правильні послідовності тегів згідно наданого і вивчених раніше зображень. Розроблена практична реалізація була запрограмована на мові Python з використанням відкритих програмних бібліотек TensorFlow та Keras.

Важливість AI для front-end розробки полягає в тому, що AI можна використовувати для автоматизації повторюваних завдань, які виникають як частина циклу front-end розробки. Використання AI зменшить навантаження на розробників, підвищуючи їх продуктивність.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Abadi M. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2016. URL: https://arxiv.org/pdf/1603.04467.pdf (дата звернення: 13.04.2022).

2. A. L. Gaunt, A probabilistic programming language for program induction. 2016. https://arxiv.org/abs/1608.04428 (дата звернення: 28.03.2022).

3. Bala Venkatesha An Introduction to Machine Learning Theory. 2018. URL: https://medium.com/@venkateshpnk22/an-introduction-to-machine-learning-theory2b09a4748187 (дата звернення: 15.04.2022).

4. Constandse C. How AI-driven website builders will change the digital landscape. 2018. URL: https://uxdesign.cc/how-aidriven-website-builders-will-change-the-digital-landscape-a5535c17bbe (дата звернення: 20.04.2022).

5. Gupta A. How Artificial Intelligence Is Reshaping Web Designing And Web Development? 2019. URL: https://www.expresscomputer.in/artificial-intelligence-ai/how-artificial-intelligence-isreshaping-web-designing-and-web-development/32201/ (дата звернення: 10.04.2022).

6. Kosmayer D. How to build a highly converting website with Artificial Intelligence. 2017. URL:https://crankwheel.com/how-to-build-a-highly-converting-website-with-artificialintelligence/ (дата звернення: 11.04.2022).

7. K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition. 2015. URL: https://arxiv.org/pdf/1409.1556.pdf (дата звернення: 05.04.2022).

8. W. Zaremba, I. Sutskever, Recurrent neural network regularization.2015. URL: https://arxiv.org/pdf/1409.2329.pdf (дата звернення: 05.04.2022).

9. Jones S. How AI is reshaping web development. Innovation Enterprise /Innovation enterprise. 2019. URL: https://channels.theinnovationenterprise.com /articles/how-ai-is-reshaping-webdevelopment (дата звернення: 15.04.2022).

10. Recurrent Neural Networks. 2014. URL: https://people.idsia.ch /~juergen/rnn.html (дата звернення: 24.04.2022).

11. Reed L. Will AI Website Builders Ever Replace Human Web Designers? Dzone. 2018. URL: https://dzone.com/articles/will-ai-websitebuilders-ever-replace- human-web-de (дата звернення: 02.04.2022).

12. Sareh Aghaei, Mohammad Ali Nematbakhsh, Hadi Khosravi Farsani Evolution of the world wide web: from web 1.0 to web 4.0. International Journal of Web & Semantic Technology. 2012. Vol. 3. №1. c. 110. URL: https://pdfs.semanticscholar.org/8cb3/93c3229e8f288febfa4dac12a0f6298efb93. pdf (дата звернення: 01.04.2022).

13. Sebastian Bassi. A Primer on Python for Life Science Researchers / PLOS Computational Biology. 2007. P. 199. (дата звернення: 20.04.2022).

14. Susloparov D. Artificial Intelligence in Web Development / Vardot. 2017. URL: https://www.vardot.com/en/blog/artificial-intelligence-web- development (дата звернення: 18.04.2022).

15. Tony Beltramelli Pix2code: Generating Code from a Graphical User Interface Screenshot. 2017. URL: https://arxiv.org/pdf/1705.07962.pdf (дата звернення: 25.03.2022).

16. Yogesh Rana. Python: Simple though an Important Programming language / International Research Journal of Engineering and Technology. 2019. P. 1856—1858. (дата звернення: 08.04.2022)

17. LSTM – мережі довготривалої короткострокової пам'яті. 2017. URL: https://habr.com/ru/company/wunderfund/blog/331310/ (дата звернення: 22.04.2022).

18. Рекурентна нейронна мережа вирішує завдання підтримання рівноваги. Методологія навчання рекурентної штучної нейронної мережі з динамічною стековою пам'яттю. 2016. URL: https://lab-music.ru/uk/rekurrentnaya-neironnaya-set-reshaet-zadachu-podderzhaniya-ravnovesiya/ (дата звернення: 24.04.2022).

19. Коваль А. Ф., Золотухін О. В. Інтелектуальний парсер для перетворення макетів web-сторінок у HTML розмітку. 2022. URL: https://nure.ua/wp-content/uploads/conf-2022-akov/telecom_2022_volume_2.pdf