
CS PROJECT
Pykemon
Snap
By,
Vignesh N

Synopsis

Pokémon, a term that has become synonymous with childhood wonder and nostalgia, represents more than just a franchise—it's a global phenomenon that has captured the hearts of millions. At its core, Pokémon are fictional creatures that inhabit a world where humans, known as Pokémon Trainers, catch and train these creatures for battles and companionship. Originating from the brilliant minds of Satoshi Tajiri and Ken Sugimori, Pokémon made its grand entrance in the 1990s, brought to life by Nintendo, Game Freak, and Creatures. The term 'Pokémon' itself is a fusion of 'Pocket Monsters,' capturing the essence of these creatures that can fit snugly into our pockets.

In the realm of pop culture, Pokémon has become an enduring phenomenon, leaving an indelible mark that spans generations. It has captured the hearts of generations, serving as a constant source of entertainment and inspiration for countless artists, writers, and creators. Recognized across diverse backgrounds, Pokémon characters, especially Pikachu with its adorable demeanor, have evolved into cultural icons. Across various forms of media such as video games, trading card games, animated TV shows, and movies, the franchise's influence is omnipresent. Beyond mere entertainment, Pokémon has permeated into art, fashion, and academic discussions, inspiring a wide range of creative expressions. More than just a pop culture sensation, Pokémon has become a cultural touchstone, fostering a sense of community among fans worldwide and symbolizing the spirit of adventure and camaraderie.

The craze surrounding Pokémon has experienced highs and lows over the years. Initially, there was an explosive rise in popularity, with kids and adults alike engrossed in the world of Pokémon. However, like any trend, there was a momentary decline. Yet, Pokémon managed to reinvent itself, finding new life with each generation of games and adaptations. The steady craze persisted, with dedicated fans continually finding new ways to engage with the franchise, ensuring its enduring appeal.

People engage with Pokémon in various ways, showcasing the franchise's versatility. Some delve into the captivating world of the anime series, following the adventures of Ash Ketchum and his trusty Pikachu. Others immerse themselves in the rich storylines of Pokémon video games, exploring vast regions, capturing creatures, and becoming Pokémon Champions. Trading Card Game (TCG) enthusiasts participate in tournaments, strategizing and competing at the highest level. Pokémon has become a diverse ecosystem, accommodating fans' interests across different mediums.

In the competitive realm of Pokémon, a sophisticated metagame has emerged. Competitive Pokémon involves strategic battles where Trainers assemble teams, each with specific Pokémon species, moves, abilities, and held items. The objective is to outwit opponents, predicting their moves and countering with precise strategies. Notably, figures like Wolfe Glick, popularly known as WolfeyVGC, have become synonymous with competitive Pokémon. They have mastered the art of battling, utilizing the depth of knowledge provided by the Pokédex to gain an advantage.

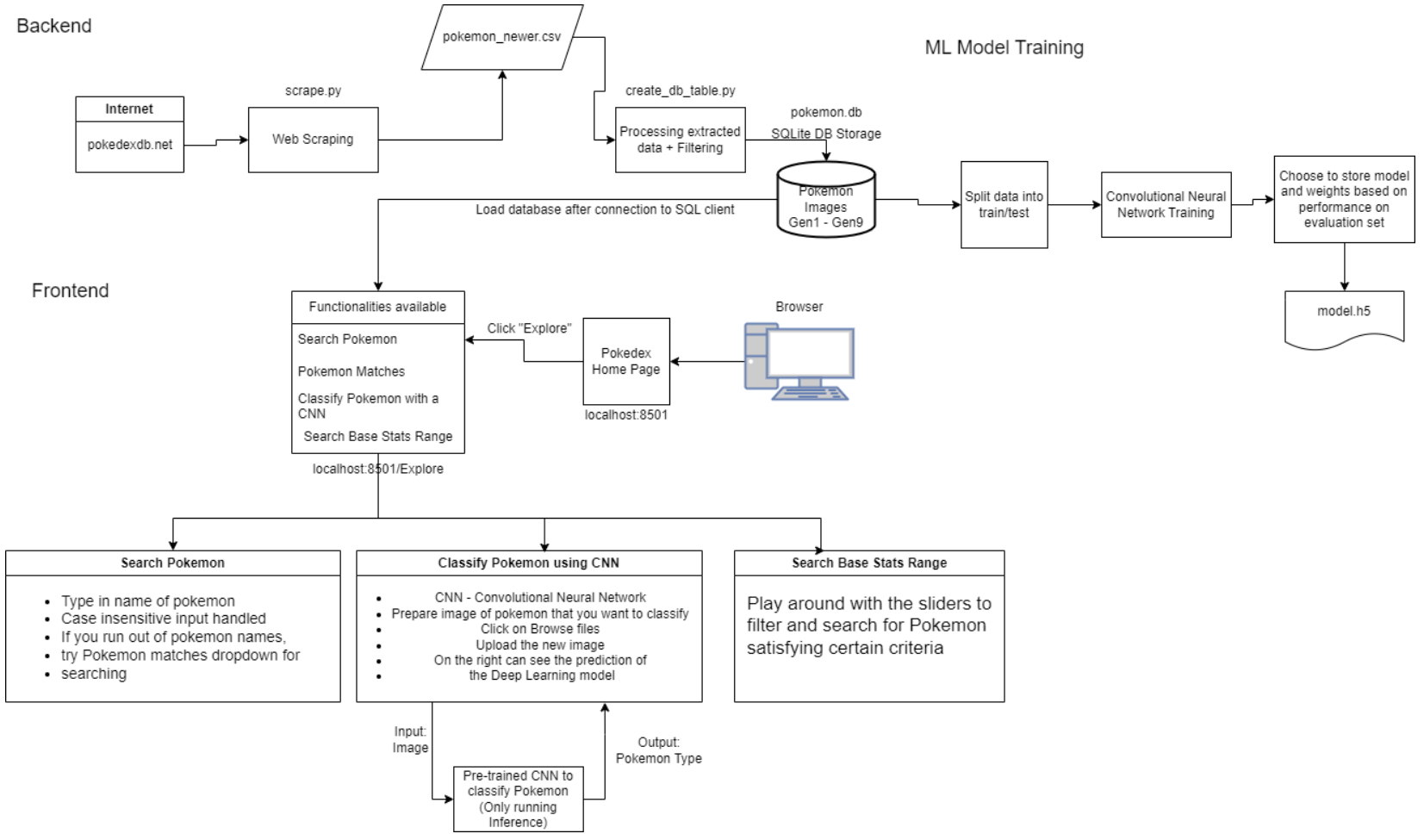
Central to the Pokémon experience is the Pokédex, a vital tool for any aspiring Trainer. It serves as an electronic encyclopedia, cataloging information about various Pokémon species. The Pokédex not only provides details about a Pokémon's characteristics, habitat, and abilities but also aids Trainers in their journey by offering valuable insights. It is an indispensable resource, guiding Trainers on their quest to catch and train Pokémon effectively.

The need for a comprehensive Pokédex is especially crucial in the competitive Pokémon scene. Competitive players meticulously research Pokémon species, moves, and abilities to craft formidable teams. My innovative application, integrating Python, SQLite, Streamlit, and machine learning models using TensorFlow and Keras, offers a revolutionary solution. By incorporating a machine learning model capable of recognizing the first generation of Pokémon, my app provides a unique advantage to competitive players. It streamlines the process of team analysis, allowing Trainers to make informed decisions swiftly. As WolfeyVGC aptly puts it, "In competitive Pokémon, a reliable Pokédex is like a sharp blade in battle; it's an essential tool for victory."

Why should people use my app? The answer is crystal clear. My app provides an unparalleled advantage to Pokémon enthusiasts, especially those in the competitive scene. Its machine learning capabilities ensure accurate and rapid Pokémon identification, saving precious time that can be better spent on strategizing and training. Additionally, it serves as an educational resource, making the vast world of Pokémon accessible to newcomers and seasoned Trainers alike, enhancing their overall experience.

In conclusion, the creation of the Pokémon encyclopedia, merging the realms of technology and nostalgia, is a testament to my passion and ingenuity. By providing a sophisticated and user-friendly application, I have not only rekindled the magic of Pokémon but also elevated it to new heights. As users explore the depths of my app, they will find themselves immersed in a world where knowledge meets strategy, and excitement intertwines with competition. With my app in hand, trainers are equipped with the ultimate tool to catch them all, making their Pokémon journey truly extraordinary. As the famous Pokémon motto goes, "Gotta catch 'em all!"

State transition diagram


Backend
pokemon_newer.csv
ML Model Training
scrape.py
create_db_table.py
Internet
pokedexdb.net
Web Scraping
Processing extracted data + Filtering
pokemon.db
SQLite DB Storage
Pokemon Images Gen1 - Gen9
Split data into train/test
Convolutional Neural Network Training
Choose to store model and weights based on performance on evaluation set
model.h5
Load database after connection to SQL client
Frontend
Functionalities available
Search Pokemon
Pokemon Matches
Classify Pokemon with a CNN
Search Base Stats Range
Click "Explore"
Pokedex Home Page
localhost:8501
Browser
localhost:8501/Explore
Search Pokemon
Type in name of pokemon
Case insensitive input handled
If you run out of pokemon names,
try Pokemon matches dropdown for
searching
Classify Pokemon using CNN
CNN - Convolutional Neural Network
Prepare image of pokemon that you want to classify
Click on Browse files
Upload the new image
On the right can see the prediction of
the Deep Learning model
Search Base Stats Range
Play around with the sliders to filter and search for Pokemon satisfying certain criteria
Input: Image
Pre-trained CNN to classify Pokemon (Only running Inference)
Output: Pokemon Type

System requirements

Hardware requirements

1. Processor (CPU): A modern multi-core processor (ex. Intel Core i5 or AMD Ryzen 5) is recommended
2. Memory (RAM): At least 8 GB of RAM is recommended
3. Storage: Preferably a SSD based storage
4. Graphics Processing Unit (GPU): Preferably a modern GPU (Nvidia GeForce or AMD Radeon)

Software requirements

1. Operating System: Can run on Windows (10 and above), MacOS (13.6.1 and above) and Ubuntu (18.04 and above)
2. Python:
    - Can use Python 3.9 and above for the entire project
    - Can use Python 3.4 and above, along Python 3.9 as a virtual environment for the entire project
3. Python Libraries
4. Web browser: Can use a recently released updated browser (ex. Chrome, Edge, Safari, Firefox or Brave)

System Design

Libraries

Library	Purpose
`streamlit`	• Web application framework used for building interactive web applications for data science and machine learning. • Used for creating the web application, managing connections to an SQLite database, and displaying various interactive com
`cv2`	• Open Source Computer Vision Library. • Used for image processing tasks.
`pickle`	Used for serializing and deserializing Python objects.
`PIL`	• Python Imaging Library. • Used for opening, manipulating, and saving image files.
`numpy`	• Numerical Python. • Used for numerical operations and array manipulations.
`plotly`	Used for creating interactive plots.
`pandas`	• Data manipulation and analysis library. • Used to read data from a CSV file into a Pandas DataFrame • Used in creating data-frames for training and testing set distributions.
`matplotlib.pylot as plt`	• Used for creating static, animated, and interactive visualizations. • Used for data visualization, particularly for plotting graphs and displaying images.
`matplotlib.image(imread)`	Used for reading images.
`sqlite3`	• C library that provides a lightweight disk-based database.

Library	Purpose
	• Used for creating a database, establishing a connection, and executing SQL queries.
`os`	Provides a way of using operating system-dependent functionality, such as reading or writing to the file system.
`shutil`	• Used for file operations, such as copying files from the training set to the test set. • Provides a higher-level interface for file operations.
`math`	• Used for mathematical operations. • Used in determining the number of rows and columns for displaying Pokémon images.
`random`	Used for generating random numbers
`datetime`	Used for working with dates and times.
`tensorflow`	• Used for building and training neural network models. • Used in loading and preprocessing images, defining a CNN model, and using callbacks for training.
`requests`	Used for sending HTTP requests to retrieve the HTML content of web pages.
`BeautifulSoup from bs4`	Used for parsing HTML content and navigating the HTML tree

Functions

Functions	Purpose
`local_css(file_name)`	Loads a local CSS file to style the display of Pokémon types.
`get_data()`	• Retrieves data from a SQLite database named 'pokemon_db' and returns it as a pandas dataframe. • Uses streamlit's `st.experimental_connection` for database access.
`get_image_path(name, id)`	Generates the path to the image file for a given Pokémon based on its name and ID.

Functions	Purpose
`display_basic_info(match)`	Displays basic information about a Pokémon, including its name, ID, image, type, height, weight, species, and abilities.
`display_base_stats_type_defenses(match)`	Displays a horizontal bar chart of the base stats of a Pokémon.
`display_radar_chart(match)`	Displays a radar chart of the base stats of a Pokémon using plotly Express.
`display_similar_pokemons(match)`	Displays information about 20 Pokémon with the most similar base stats to the selected Pokémon.
`hide_streamlit_style`	Defines a CSS style to hide streamlit's default menu and footer.
`st.set_page_config`	Configures the settings for the Streamlit app's page.
`st.cache_resource`	• Decorator for caching resources (ex. files, data) to improve app performance. • Decorates the `get_data` function to cache the result, which is fetching data from an SQLite database.
`st.experimental_connection`	• Creates a connection to an SQLite database using streamlit's experimental connection feature. • Used in the `get_data` function to establish a connection to the 'pokemon_db' SQLite database.
`st.sidebar.title`	Sets the title of the sidebar in the streamlit app.
`st.sidebar.text_input`	• Displays a text input box in the sidebar for user interaction. • Used to input the Pokémon name for searching.
`st.sidebar.selectbox`	• Displays a select box in the sidebar for user interaction. • Used to select a Pokémon name from the dropdown menu.
`st.sidebar.form`	• Creates a form in the sidebar for user interaction. • Used for setting up a form for searching Pokémon based on base stats.

Functions	Purpose
`st.form_submit_button`	• Renders a submit button for a form. • Used to create a button that, when pressed, submits the form for searching Pokémon based on base stats.
`st.image`	• Displays an image in the streamlit app. • Used to display Pokémon images
`st.subheader`	• Displays a subheader in the streamlit app. • Used to display subheaders for Pokémon abilities and output class.
`st.metric`	• Displays a metric in the streamlit app. • Used to display Pokémon height and weight as metrics.
`st.button`	• Displays a button in the streamlit app. • Used for creating a button to trigger the search for similar Pokémon.
`st.table`	• Displays a table in the streamlit app. • Used to display tables of Pokémon data.
`st.markdown`	• Renders Markdown-formatted text in the streamlit app. • Used for rendering HTML and providing information in Markdown format.
`st.plotly_chart`	• Displays a plotly chart in the streamlit app. • Used to display radar charts and other interactive charts.
`st.write`	• Writes text or data to the streamlit app. • Used for writing various information and messages to the app
`st.experimental_rerun`	• Enables rerunning of the streamlit app. • Used to rerun the app when specific conditions are met.
`pd.read_csv`	• Reads a CSV file into a pandas dataframe. • The CSV file contains information about Pokémon.
`to_dict`	Converts the pandas data-frame to a dictionary.
`os.path.exists`	Checks if a file or directory exists
`st.connection`	Creates a connection to an SQLite database using streamlit's connection feature.

Functions	Purpose
`with conn.session as s`	• Utilizes the connection session to execute SQL commands within a block. • Creates an SQLite table named 'pokemon' if it doesn't exist.
`s.execute`	• Executes an SQL `INSERT` statement to insert Pokémon data into the 'pokemon' table. • The SQL statement includes placeholders (`:national_no` , `:name` , etc.) that are filled with values from the current record in the loop.
`s.commit`	• Commits the changes made within the session to the SQLite database. • Persists the data in the 'pokemon' table.
`s.close`	• Closes the connection session. • Releases resources associated with the SQLite connection.
`prep_test_data`	Copies 15 random images from the train folders to the test folders for a given pokemon class
`show_pokemon`	Displays a grid of pokemon images.
`cnn`	Defines a Convolutional Neural Network (CNN) model using keras.
`os.listdir(directory)`	Returns a list containing the names of the entries in the specified directory.
`shutil.copy(src, dst)`	Copies the contents of the source file or directory to the destination directory.
`random.sample:`	Returns a specified number of unique elements selected randomly from a sequence.
`pickle.dump(obj, file, protocol)`	Writes a pickled representation of an object to a file.
`image.load_img(path)`	Loads an image from file path into PIL format.
`ImageDataGenerator(rescale, shear_range, zoom_range, horizontal_flip)`	Generates batches of augmented data for training neural networks.
`datagen.flow_from_directory(directory, target_size, batch_size, class_mode, color_mode)`	Takes the path to a directory and generates batches of augmented data (used for training and validation sets).
`neuralnetwork_cnn.fit_generator(generator, validation_data, callbacks, epochs)`	• Fits the model on data generated batch-by-batch by a Python generator.

Functions	Purpose
	• Trains the model on data generated batch by batch by a Python generator.
`pd.DataFrame`	Creates a DataFrame from a dictionary or a list of dictionaries.
`neuralnetwork_cnn.summary`	Prints a string summary of the neural network.
`plot_model(model, show_shapes)`	• Plots a graphical representation of the Keras model. • Plot the model architecture in a graphical form.
`plt.subplots(figsize)`	Create a figure and a set of subplots with a specified size
`Sequential()`	Linear stack of layers for building a neural network model.
`ModelCheckpoint(filepath, monitor, verbose, save_best_only, mode)`	Callbacks to save the model after every epoch.
`ReduceLROnPlateau(monitor, patience, verbose)`	Reduces the learning rate when a metric has stopped improving.
`requests.get(url)`	Sends an HTTP GET request to the specified URL and returns a `Response` object.
`BeautifulSoup(response.content, 'html_parser')`	Initializes a BeautifulSoup object to parse HTML content.
`find(tag/class)`	Finds the first occurrence of a specified HTML tag or class within the HTML content.
`find_all()`	Finds all occurrences of a specified HTML tag or class within the HTML content.
`get_text()`	Gets the text content of an HTML element.
`to_csv('filename.csv')`	Writes a data-frame to a CSV file.

Data dictionary

Data file	Purpose
pokemon_newer.csv	Acts as a convenient and intermediary representation of the pokemons' data during the transition between different stages of the data processing pipeline.

ER diagram

Name	Type	Schema
pokemon		CREATE TABLE pokemon (National_no TEXT, Name TEXT, type_number NUM, type_1 TEXT, type_2 TEXT, Total NUM, HP NUM, Attack NUM, Defense NUM, Special_attack NUM, Special_defense NUM, Speed NUM, Species TEXT, Height TEXT, Weight TEXT, abilities_number NUM, ability_1 TEXT, ability_2 TEXT, ability_hidden)
National_no	TEXT	"National_no" TEXT
Name	TEXT	"Name" TEXT
type_number	NUM	"type_number" NUM
type_1	TEXT	"type_1" TEXT
type_2	TEXT	"type_2" TEXT
Total	NUM	"Total" NUM
HP	NUM	"HP" NUM
Attack	NUM	"Attack" NUM
Defense	NUM	"Defense" NUM
Special_attack	NUM	"Special_attack" NUM
Special_defense	NUM	"Special_defense" NUM
Speed	NUM	"Speed" NUM
Species	TEXT	"Species" TEXT
Height	TEXT	"Height" TEXT
Weight	TEXT	"Weight" TEXT
abilities_number	NUM	"abilities_number" NUM
ability_1	TEXT	"ability_1" TEXT
ability_2	TEXT	"ability_2" TEXT
ability_hidden		"ability_hidden"

Source code

1_Explore.py

```
import streamlit as st
import cv2
import pickle
from PIL import Image
import numpy as np
import plotly.express as px
import pandas as pd
import matplotlib.pyplot as plt
import sqlite3
import os
from tensorflow.keras.models import load_model

st.set_page_config(page_title = "Pokédex", layout = "wide")

# css file for displaying Pokemon type (fire, water etc.)
def local_css(file_name):
    with open(file_name) as f:
        st.markdown(f'<style>{f.read()}</style>', unsafe_allow_html=True)

@st.cache_resource
def get_data():
    conn = st.experimental_connection('pokemon_db', type='sql')
    with conn.session as s:
        # Query and display the data you inserted
        df = pd.DataFrame(conn.query('select * from pokemon'))
        s.close()
    return df

# load css file and get data
local_css('style.css')
df = get_data()

# sidebar configuration for searching Pokemon by name
st.sidebar.title('Pokédex')
name = st.sidebar.text_input('Search Name', '').lower() # input name
# find names that matches input and return it in a list
matches = list(df[df['Name'].str.lower().str.contains(name)]['Name'])
# dropdown menu with names that matches input
if len(matches) >= 1:
    name = st.sidebar.selectbox('Pokemon Matches', matches).lower()
else: # if no name matches input
    name = st.sidebar.selectbox('Pokemon Matches', ['No match'])
```

```
# filter row of data that matches Pokemon selected in dropdown menu
match = df[df['Name'].str.lower() == name]

# select information to view
info_list = ['Basic Information', 'Base Stats & Type Defenses', 'Radar Chart']
selected_info = st.sidebar.multiselect('View Information', info_list,
default = info_list)

model = load_model('model.h5')
uploaded_file = st.sidebar.file_uploader("Choose a image file",type=["jpg","png"])

if uploaded_file is not None:
    # Convert the file to an opencv image.
    file_bytes = np.asarray(bytearray(uploaded_file.read()), dtype=np.uint8)
    test_img = cv2.imdecode(file_bytes, 1)

    test_img = cv2.cvtColor(test_img, cv2.COLOR_RGB2BGR)
    test_img = cv2.resize(test_img, (64,64))
    #plt.imshow(test_img)
    test_img = test_img[np.newaxis,...]

    output = model.predict(test_img)
    output_index = np.argmax(output)
    with open('target_classes.pickle', 'rb') as handle:
        target_classes = pickle.load(handle)
    output_name = target_classes[output_index][1]
    left_co, cent_co,last_co = st.columns(3)
    with cent_co:
        st.image(test_img, width=300)
        st.subheader(f"Output class: {output_name}")

# search Pokemon using min and max base stats (speed, special defense etc.)
with st.sidebar.form(key="my_form"):
    st.subheader('Search Base Stats Range')
    min_speed, max_speed = st.select_slider('Speed', range(251), value = [0, 250])
    min_sp_def, max_sp_def = st.select_slider('Special Defense', range(251), value
     = [0, 250])
    min_sp_atk, max_sp_atk = st.select_slider('Special Attack', range(251), value
     = [0, 250])
    min_def, max_def = st.select_slider('Defense', range(251), value = [0, 250])
    min_atk, max_atk = st.select_slider('Attack', range(251), value = [0, 250])
    min_hp, max_hp = st.select_slider('HP', range(251), value = [0, 250])

    # pressed is a Boolean that becomes True when the "Search Pokemon" button is
     pressed
    # code to handle button pressing is all the way at the bottom
    pressed = st.form_submit_button("Search Pokemon")

# display credits on sidebar
st.sidebar.subheader('Info')
st.sidebar.markdown('Pokemon images taken from <a href="https://www.kaggle.com/
```

```
    datasets/kvpratama/pokemon-images-dataset">this Kaggle link</a>.',
  unsafe_allow_html = True)

# use Pokemon name and id to get image path, refer to 'pokemon_images' folder to
 see how images are named
def get_image_path(name, id):
    if 'Mega' in name:
        if name.endswith(' X'):
            path = 'pokemon_images/' + str(id) + '-mega-x.png'
        elif name.endswith(' Y'):
            path = 'pokemon_images/' + str(id) + '-mega-y.png'
        else:
            path = 'pokemon_images/' + str(id) + '-mega.png'
    elif 'Rotom' in name:
        rotom_type = name.split()[0].lower()
        path = 'pokemon_images/' + str(id) + '-' + rotom_type + '.png'
    elif 'Forme' in name or 'Cloak' in name  or 'Form' in name:
        if 'Zygarde' in name: # only 1 image present for Zygarde
            path = 'pokemon_images/' + str(id) + '.png'
        else:
            type = name.split()[1].lower()
            path = 'pokemon_images/' + str(id) + '-' + type + '.png'
    elif 'Primal' in name:
        type = name.split()[0].lower()
        path = 'pokemon_images/' + str(id) + '-' + type + '.png'
    elif 'Aceus' in name:
        path = 'pokemon_images/' + str(id) + '-normal.png'
    else:
        path = 'pokemon_images/' + str(id) + '.png'
    return path


def display_basic_info(match):
    # get basic info data
    name = match['Name'].iloc[0]
    id = match['National_no'].iloc[0]
    height = str(match['Height'].iloc[0])
    weight = str(match['Weight'].iloc[0])
    species = ' '.join(match['Species'].iloc[0].split(' ')[:-1])
    type1 = match['type_1'].iloc[0]
    type2 = match['type_2'].iloc[0]
    type_number = match['type_number'].iloc[0]
    ability1 = match['ability_1'].iloc[0]
    ability2 = match['ability_2'].iloc[0]
    ability_hidden = match['ability_hidden'].iloc[0]

    st.title(name + ' #' + str(id).zfill(3))
    col1, col2, col3 = st.columns(3)

    # leftmost column col1 displays pokemon image
    try:
```

```
        path = get_image_path(name, id)
        image = Image.open(path)
        col1.image(image)
    except:
        col1.write('Image not available.')

    # middle column col2 displays nicely formatted Pokemon type using css loaded
             earlier
    with col2.container():
        col2.write('Type')
        # html code that loads the class defined in css, each Pokemon type has a
                    different style color
        type_text = f'<span class="icon type-{type1.lower()}">{type1}</span>'
        if type_number == 2:
            type_text += f'<span class="icon type-{type2.lower()}">{type2}</span>'
        # markdown displays html code directly
        col2.markdown(type_text, unsafe_allow_html=True)
        col2.metric("Height", height + " m")
        col2.metric("Weight", weight + " kg")

    # rightmost column col3 displays Pokemon abilities
    with col3.container():
        col3.metric("Species", species)
        col3.write('Abilities')
        # print(ability1, type(ability1))
        # print(ability2, type(ability2))
        # print(ability_hidden, type(ability_hidden))
        if ability1 != '' and ability1 != None:
            col3.subheader(ability1)
        if ability2 != '' and ability2 != None:
            col3.subheader(ability2)
        if ability_hidden != '' and ability_hidden != None:
            col3.subheader(ability_hidden + ' (Hidden)')

def display_base_stats_type_defenses(match):
    with st.container():
        col1, col2 = st.columns(2)

        # left column col1 displays horizontal bar chart of base stats
        col1.subheader('Base Stats')
        # get base stats of Pokemon and rename columns nicely
        df_stats = match[['HP', 'Attack', 'Defense', 'Special_attack',
                 'Special_defense', 'Speed']]
        df_stats = df_stats.T
        df_stats.columns=['stats']

        # plot horizontal bar chart using matplotlib.pyplot
        fig, ax = plt.subplots()
        ax.barh(y = df_stats.index, width = df_stats.stats)
        plt.xlim([0, 250])
        col1.pyplot(fig)
```

```
def display_radar_chart(match):
    st.header('Radar Chart of Base Stats')

    # get base stats of Pokemon and rename columns nicely
    df_stats = match[['HP', 'Attack', 'Defense', 'Special_attack',
         'Special_defense', 'Speed']]
    df_stats = df_stats.T
    df_stats.columns=['stats']

    # use plotly express to plot out radar char of stats
    fig = px.line_polar(df_stats, r='stats', theta=df_stats.index,
     line_close=True ,range_r=[0, 250])
    st.plotly_chart(fig)

    if st.button('Search for Pokemons with Similar Base Stats'):
        display_similar_pokemons(match)

def display_similar_pokemons(match):
    # get base stats of Pokemon and rename columns nicely
    df_stats = match[['HP', 'Attack', 'Defense', 'Special_attack',
         'Special_defense', 'Speed']]

    # get stats of all other Pokemon in the full dataframe
    df_stats_all = df[['Name', 'HP', 'Attack', 'Defense', 'Special_attack',
        'Special_defense', 'Speed']].set_index('Name')

    # find difference between stat of Pokemon and each of the other Pokemons
    diff_df = pd.DataFrame(df_stats_all.values - df_stats.values,
         index = df_stats_all.index)

    # find norm 'distance' between this Pokemon and all other Pokemon
       norm_df = diff_df.apply(np.linalg.norm, axis=1)
    # find 20 other Pokemon with smallest distance, i.e. with most similar base
       stats to this Pokemon
    similar_pokemons = norm_df.nsmallest(21)[1:22].index
    # store all similar Pokemon with their stats in df
    similar_pokemons_df = df_stats_all.loc[similar_pokemons]
    st.write(similar_pokemons_df)
    # display name, image, radar chart of each similar Pokemon
    for row in similar_pokemons_df.iterrows():
        name = row[0]
        st.subheader(name) # display Pokemon name
        id = df[df.Name == name]['National_no'].iloc[0]
        # display Pokemon image
        try:
            path = get_image_path(name, id)
            image = Image.open(path)
            st.image(image)
        except:
            st.write('Image not available.')
```

```
        # display radar chart
        fig = px.line_polar(row[1], r=name, theta=row[1].index, line_close=True,
         range_r=[0, 255])
        st.plotly_chart(fig)

    # display full table of all 20 similar Pokemons and their stats
    st.subheader('20 Most Similar Pokemons')
    st.table(similar_pokemons_df)

# if "Search Pokemon" button is not pressed,
# i.e. we are searching Pokemon by name instead of by base stats slider section
if not pressed:
    if len(match) == 0:
        st.write('Enter name to search for details.')

    # display information of Pokemon according to the information the user selects
       in the sidebar multiselector
    elif len(match) == 1:
        if 'Basic Information' in selected_info:
            display_basic_info(match)
        if 'Base Stats & Type Defenses' in selected_info:
            display_base_stats_type_defenses(match)
        if 'Radar Chart' in selected_info:
            display_radar_chart(match)

# if "Search Pokemon" button IS PRESSED,
# filter Pokemon according to the base stats in the slider
# display all the matched Pokemon and their stats in a table
# we do not display every bit of information about the search Pokemons as the list
   may be huge and the app can hang

else:
    # get base stats of all Pokemon
    df_stats_all = df[['Name', 'HP', 'Attack', 'Defense', 'Special_attack',
         'Special_defense', 'Speed']].set_index('Name')

    # filter stats according to search criteria from the sliders
    searched_pokemons_df = df_stats_all[
        (df_stats_all['HP'] >= min_hp) &
          (df_stats_all['HP'] <= max_hp) &
        (df_stats_all['Attack'] >= min_atk) &
          (df_stats_all['Attack'] <= max_atk) &
        (df_stats_all['Defense'] >= min_def) &
          (df_stats_all['Defense'] <= max_def) &
        (df_stats_all['Special_attack'] >= min_sp_atk) &
          (df_stats_all['Special_attack'] <= max_sp_atk) &
        (df_stats_all['Special_defense'] >= min_sp_def) &
          (df_stats_all['Special_defense'] <= max_sp_def) &
        (df_stats_all['Speed'] >= min_speed) &
          (df_stats_all['Speed'] <= max_speed)
```

```
    ]
    st.header('Pokemon Search Using Base Stats')
    st.table(searched_pokemons_df)

hide_streamlit_style = """
<style>
#MainMenu {visibility: hidden;}
footer {visibility: hidden;}
</style>
"""

st.markdown(hide_streamlit_style, unsafe_allow_html=True)
```

create_db_table.py

```
import pandas as pd
import streamlit as st
import os

if not os.path.exists('pokemon.db'):
    pokemon_db_csv_dict =
      pd.read_csv('pokedex_newer.csv').to_dict(orient='records')
    # Create the SQL connection to db as specified in your secrets file.
    conn = st.connection('pokemon_db', type='sql')
    # Insert some data with conn.session.
    with conn.session as s:
        s.execute('CREATE TABLE IF NOT EXISTS pokemon \
                (National_no TEXT,\
                Name TEXT,\
                type_number NUM,\
                type_1 TEXT,\
                type_2 TEXT,\
                Total NUM,\
                HP NUM,\
                Attack NUM,\
                Defense NUM,\
                Special_attack NUM,\
                Special_defense NUM,\
                Speed NUM,\
                Species TEXT,\
                Height TEXT,\
                Weight TEXT,\
                abilities_number NUM,\
                ability_1 TEXT,\
                ability_2 TEXT,\
                ability_hidden);')

        for record in pokemon_db_csv_dict:
            s.execute(
                'INSERT INTO pokemon (National_no, Name, type_number, type_1,
                 type_2, Total, HP, Attack, Defense, Special_attack,
```

```
                 Special_defense, Speed, Species, Height, Weight, abilities_number
                 ability_1, ability_2, ability_hidden) VALUES (:national_no, :name
                 :type_number, :type_1, :type_2, :total, :hp, :attack, :defense,
                 :sp_attack, :sp_defense, :species, :height, :weight,
                 :abilities_number, :ability_1, :ability_2, :ability_hidden);',
                params=
                dict(
                    national_no = record['National_no'],
                    name = record['Name'],
                    type_number = record['type_number'],
                    type_1 = record['type_1'],
                    type_2 = record['type_2'],
                    total = record['Total'],
                    hp = record['HP'],
                    attack = record['Attack'],
                    defense = record['Defense'],
                    sp_attack = record['Special_attack'],
                    sp_defense = record['Special_defense'],
                    speed = record['Speed'],
                    species = record['Species'],
                    height = record['Height'],
                    weight = record['Weight'],
                    abilities_number = record['abilities_number'],
                    ability_1 = record['ability_1'],
                    ability_2 = record['ability_2'],
                    ability_hidden = record['ability_hidden'],
                    )
            )
        s.commit()
        s.close()
```

model_training.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.image import imread
import random, datetime, os, shutil, math

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image

dir_path='/content/gdrive/MyDrive/Kaggle/PokemonData_train'
classes = os.listdir(dir_path)
len(classes)

train_dir='/content/gdrive/MyDrive/Kaggle/PokemonData_train'
test_dir='/content/gdrive/MyDrive/Kaggle/PokemonData_test'

# Copying 15 random images from train folders to test folders
def prep_test_data(pokemon, train_dir, test_dir):
```

```
  pop = os.listdir(train_dir+'/'+pokemon)
  test_data=random.sample(pop, 15)
  print(test_data)
  for f in test_data:
    shutil.copy(train_dir+'/'+pokemon+'/'+f, test_dir+'/'+pokemon+'/')

#performing samething for each folder in train folder
for poke in os.listdir(train_dir):
  prep_test_data(poke, train_dir, test_dir)

target_classes = os.listdir(train_dir)
num_classes = len(target_classes)
print('Number of target classes:', num_classes)
print(list(enumerate(target_classes)))
import pickle

with open('target_classes.pickle', 'wb') as handle:
    pickle.dump(list(enumerate(target_classes)), handle,
         protocol=pickle.HIGHEST_PROTOCOL)

training_set_distribution = [len(os.listdir(os.path.join(train_dir, dir))) for
 dir in os.listdir(train_dir)]
testing_set_distribution = [len(os.listdir(os.path.join(test_dir, dir))) for
 dir in os.listdir(test_dir)]

def show_pokemon(pokemon):
  num = len(pokemon)
  if num == 0:
    return None
  rows = int(math.sqrt(num))
  cols = (num+1)//rows
  f, axs = plt.subplots(rows, cols)
  fig = 0
  for b in pokemon:
    img = image.load_img(b)
    row = fig // cols
    col = fig % cols
    axs[row, col].imshow(img)
    fig += 1
  plt.show()

dir_name = os.path.join(train_dir,"Ditto")
all_images = [os.path.join(dir_name, fname) for fname in os.listdir(dir_name)]
show_pokemon(all_images[:6])

image_size = (64, 64, 3)
datagen=ImageDataGenerator(rescale = 1./255,
                           shear_range=0.2,
                           zoom_range=0.2,
                           horizontal_flip=True,
                           )
```

```
training_set=datagen.flow_from_directory(train_dir,
                                         target_size=image_size[:2],
                                         batch_size=32,
                                         class_mode='categorical',
                                         color_mode='rgb'
                                         )

validation_set=datagen.flow_from_directory(test_dir,
                                           target_size=image_size[:2],
                                           batch_size=32,
                                           class_mode='categorical',
                                           color_mode='rgb'
                                           )

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
    GlobalAveragePooling2D
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from keras.models import Sequential
from keras import applications
from keras.utils import plot_model

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=7)
filepath = "model.h5"
ckpt = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True,
    mode='min')
rlp = ReduceLROnPlateau(monitor='loss', patience=3, verbose=1)

#defining model
def cnn(image_size, num_classes):
    classifier = Sequential()
    classifier.add(Conv2D(64, (5, 5), input_shape=image_size, activation='relu',
      padding='same'))
    classifier.add(MaxPooling2D(pool_size = (2, 2)))
    classifier.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    classifier.add(MaxPooling2D(pool_size = (2, 2)))
    classifier.add(Flatten())
    classifier.add(Dense(num_classes, activation = 'softmax'))
    classifier.compile(optimizer='adam', loss='categorical_crossentropy',
      metrics=['acc'])
    return classifier

neuralnetwork_cnn = cnn(image_size, num_classes)
neuralnetwork_cnn.summary()
plot_model(neuralnetwork_cnn, show_shapes=True)

history = neuralnetwork_cnn.fit_generator(
    generator=training_set, validation_data=validation_set,
    callbacks=[es, ckpt, rlp], epochs = 20,
)
```

```
fig, ax = plt.subplots(figsize=(20, 6))
pd.DataFrame(history.history).iloc[:, :-1].plot(ax=ax)
```

Pokedex.py

```
import streamlit as st

st.set_page_config(
    page_title="Pokedex",
    # page_icon="",
)

page_bg_img = '''
<style>
.stApp {
background-image: url("https://images.unsplash.com/photo-1638613067237-b1127ef06c0
 0?auto=format&fit=crop&q=80&w=2982&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYW
 dlfHx8fGVufDB8fHx8fA%3D%3D");
background-size: 100%;
background-repeat: no-repeat;
background-position: center;

}
</style>
'''

st.markdown(page_bg_img, unsafe_allow_html=True)
```

scrape.py

```
import requests
from bs4 import BeautifulSoup
import numpy as np
import pandas as pd

base = "https://pokemondb.net"
url = "https://pokemondb.net/pokedex/all"
response = requests.get(url)
soup = BeautifulSoup(response.content, 'html.parser')

header = soup.find('h1')
print(header)
#obtain webpage intro
intro = soup.find('div', attrs={"class": "panel panel-intro"})
print(intro)

mapping_pokemon_all_table = {
    0: 'National_no',
    1: 'Name',
    2: 'Types',
    3: 'Total',
```

```
    4: 'HP',
    5: 'Attack',
    6: 'Defense',
    7: 'Special_attack',
    8: 'Special_defense',
    9: 'Speed',
}

mapping_vitals_table = {
    2: 'Species',
    3: 'Height',
    4: 'Weight',
    5: 'Abilities'
}

#not using special words currently
special_words = ['Mega', 'Alolan', 'Partner', 'Galarian', 'Hisuian', 'Combat',
 'Blaze', 'Aqua', 'Paldean', 'Sunny', 'Rainy', 'Snowy', 'Primal', 'Normal',
 'Attack', 'Defense', 'Speed', 'Plant', 'Sandy', 'Tra']

pokedex = soup.find('table', id="pokedex").find_all('tr')

pokemon_all_table = []
for i in range(1,len(pokedex)):
    temp = pokedex[i]
    record = dict()
    for index, colname in mapping_pokemon_all_table.items():
        record[colname] = temp.find_all('td')[index].get_text()
    link = base+temp.find_all('a', href=True)[0]['href']
    # print(link)
    sub_response = requests.get(link)
    sub_soup = BeautifulSoup(sub_response.content, 'html.parser')
    vitals_table = sub_soup.find('table', class_="vitals-table").find_all('tr')
    for sub_index, sub_colname in mapping_vitals_table.items():
        record[sub_colname] = vitals_table[sub_index].find('td').get_text()
    pokemon_all_table.append(record)


pokedf = pd.DataFrame(pokemon_all_table)
pokedf.to_csv('pokedex_new.csv')
```

style.css

```
.icon{
    display:inline-block;
    width:66px;
    margin-bottom:4px;
    background:#dbdbdb;
    border:1px solid #a3a3a3;
    border-radius:4px;
    border:1px solid rgba(0,0,0,.2);
```

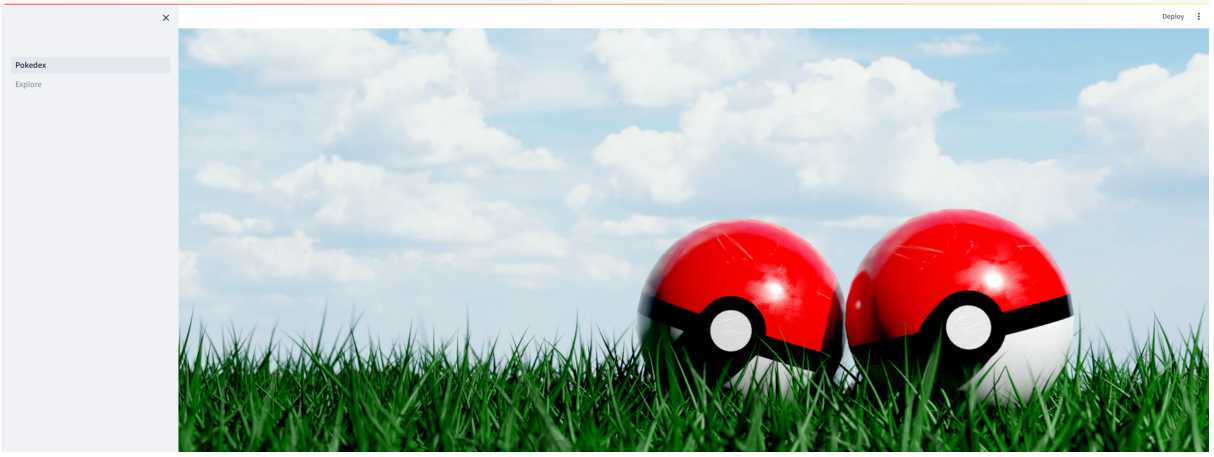
```
    color:#fff;
    font-size:.75rem;
    font-weight:normal;
    line-height:1.5rem;
    text-align:center;
    text-shadow:1px 1px 2px rgba(0,0,0,.7);
    text-transform:uppercase;
    transition:opacity .4s
}

.type-normal{
    background-color:#aa9
}
.type-fire{
    background-color:#f42
}
.type-water{
    background-color:#39f
}
.type-electric{
    background-color:#fc3
}
.type-grass{
    background-color:#7c5
}
.type-ice{
    background-color:#6cf
}
.type-fighting{
    background-color:#b54
}
.type-poison{
    background-color:#a59
}
.type-ground{
    background-color:#db5
}
.type-flying{
    background-color:#89f
}
.type-psychic{
    background-color:#f59
}
.type-bug{
    background-color:#ab2
}
.type-rock{
    background-color:#ba6
}
.type-ghost{
    background-color:#66b
```

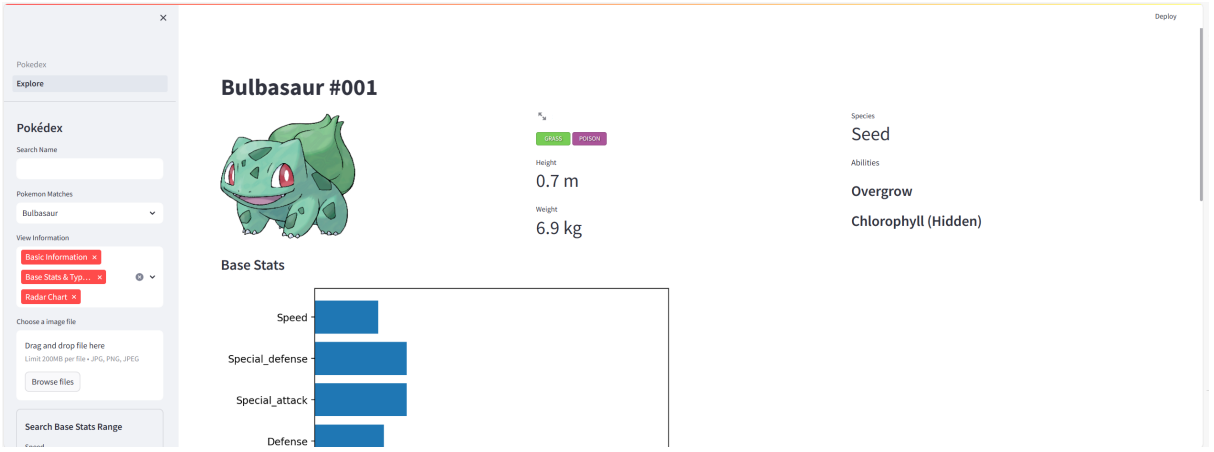
```
}
.type-dragon{
    background-color:#76e
}
.type-dark{
    background-color:#754
}
.type-steel{
    background-color:#aab
}
.type-fairy{
    background-color:#e9e
}
.type-curse{
    background-color:#698
}
```

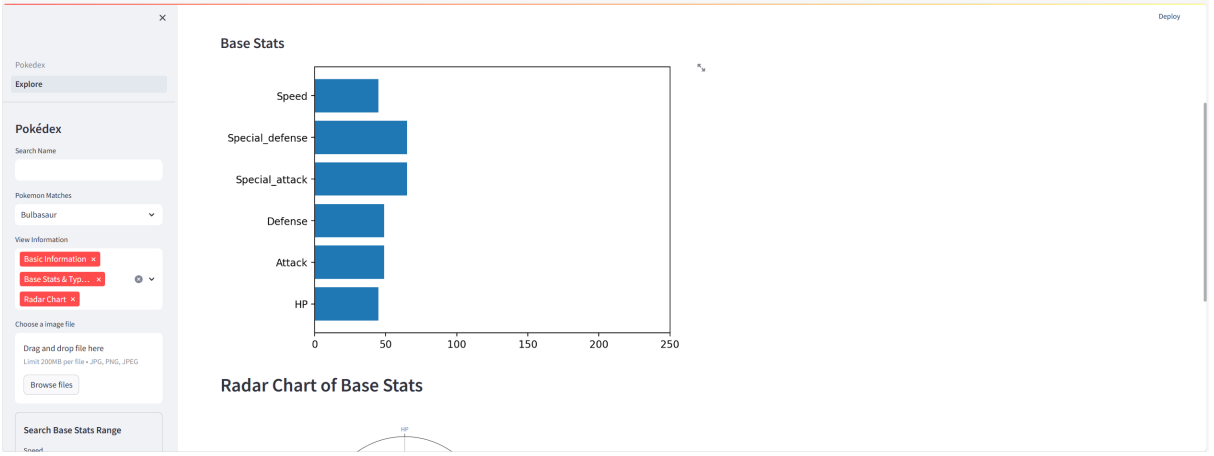
secrets.toml

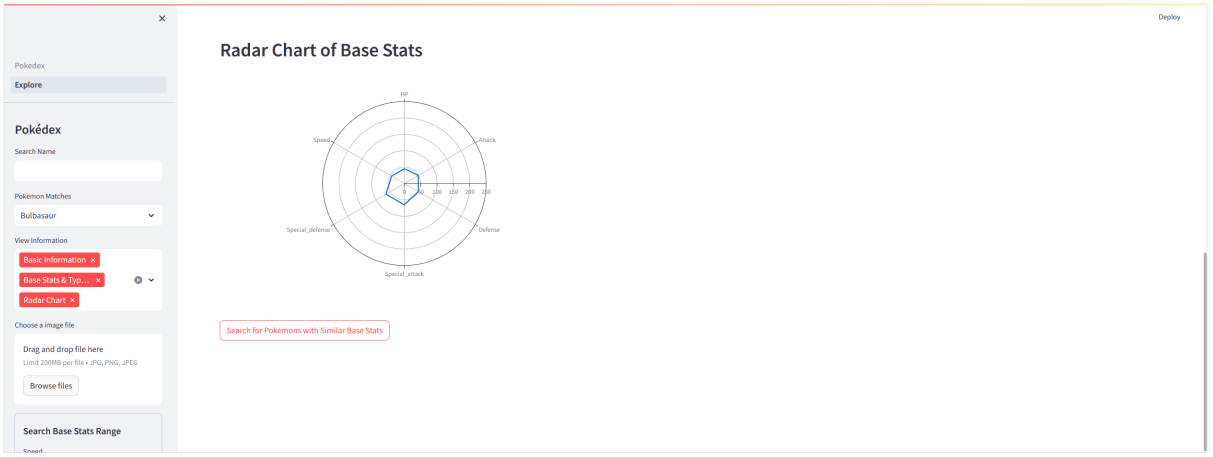
```
[connections.pokemon_db]
url = "sqlite:///pokemon.db"
```

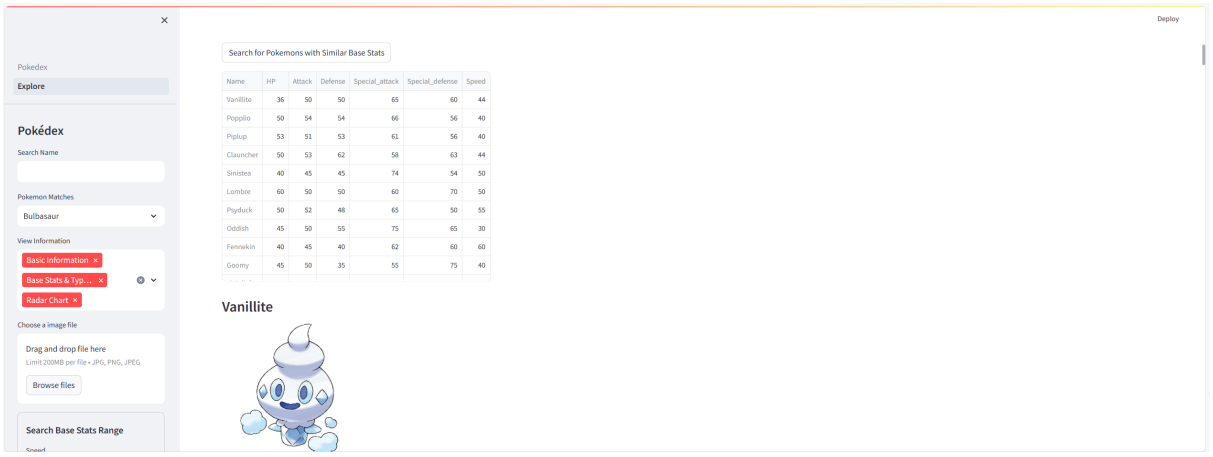
Output

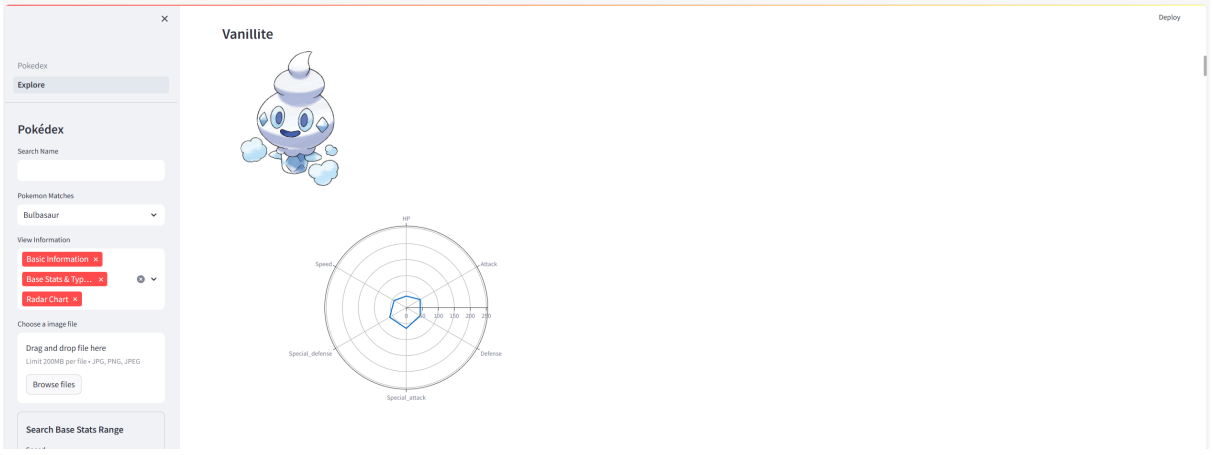

Deploy
Pokedex
Explore

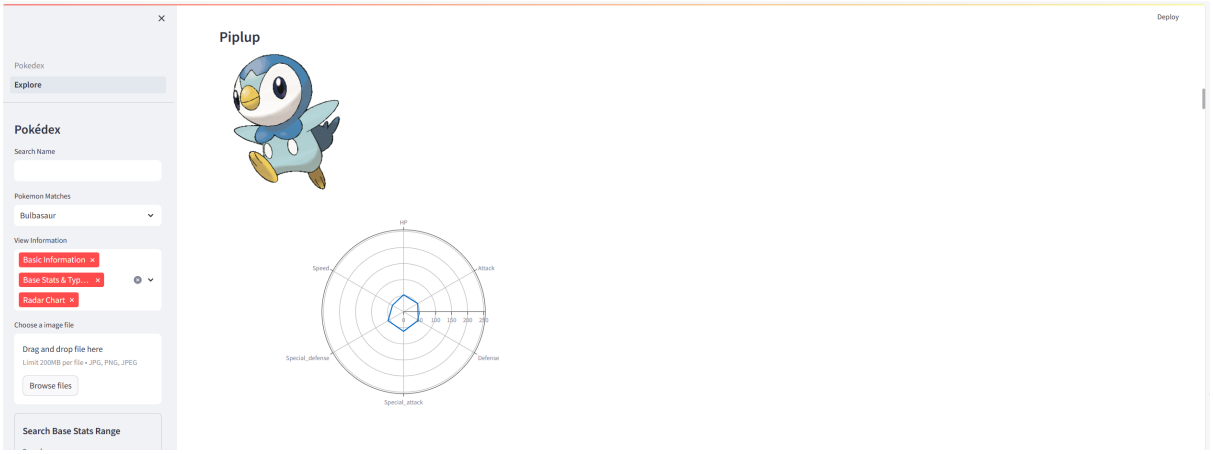

Deploy
Pokedex
Explore
Pokédex
Search Name
Pokemon Matches
Bulbasaur
View Information
Basic Information
Base Stats & Typ...
Radar Chart
Choose a image file
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
Search Base Stats Range
Bulbasaur #001
GRASS
POISON
Height
0.7 m
Weight
6.9 kg
Species
Seed
Abilities
Overgrow
Chlorophyll (Hidden)
Base Stats
Speed
Special_defense
Special_attack
Defense

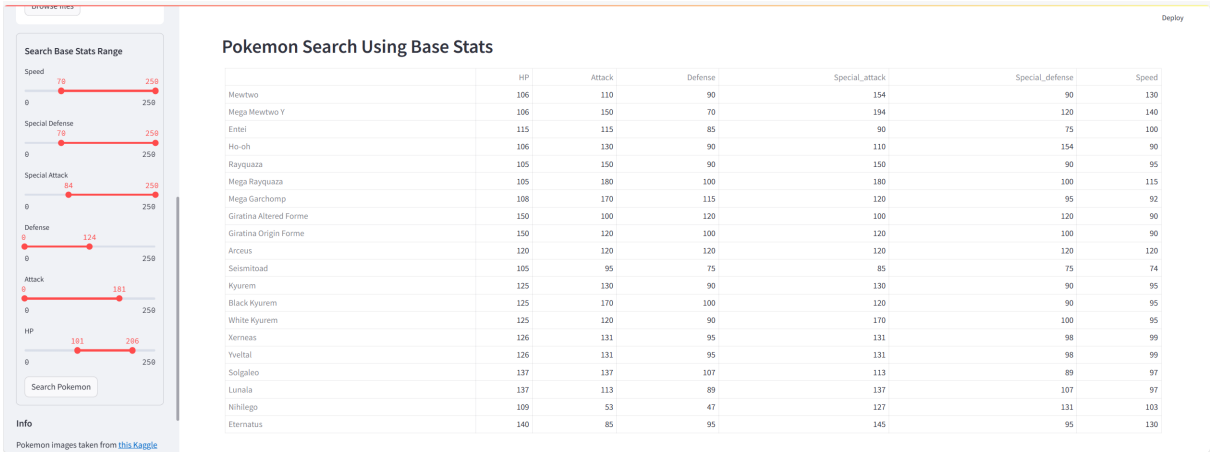

Deploy
Pokedex
Explore
Pokédex
Search Name
Pokemon Matches
Bulbasaur
View Information
Basic Information
Base Stats & Typ...
Radar Chart
Choose a image file
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
Search Base Stats Range
Base Stats
Speed
Special_defense
Special_attack
Defense
Attack
HP
0
50
100
150
200
250
Radar Chart of Base Stats


Deploy
Pokedex
Explore
Pokédex
Search Name
Pokemon Matches
Bulbasaur
View Information
Basic Information
Base Stats & Typ...
Radar Chart
Choose a image file
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
Search Base Stats Range
Radar Chart of Base Stats
HP
Attack
Defense
Special_attack
Special_defense
Speed
Search for Pokemons with Similar Base Stats

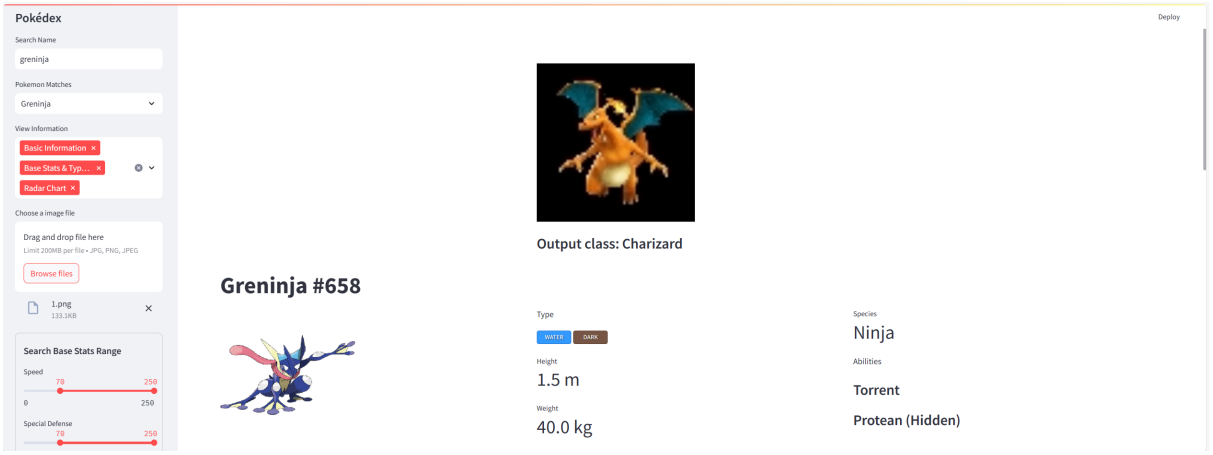

Deploy
Pokedex
Explore
Pokédex
Search Name
Pokemon Matches
Bulbasaur
View Information
Basic Information
Base Stats & Typ...
Radar Chart
Choose a image file
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
Search Base Stats Range
Search for Pokemons with Similar Base Stats
Name HP Attack Defense Special_attack Special_defense Speed
Vanillite 36 50 50 65 60 44
Popplio 50 54 54 66 56 40
Piplup 53 51 53 61 56 40
Clauncher 50 53 62 58 63 44
Sinistea 40 45 45 74 54 50
Lombre 60 50 50 60 70 50
Psyduck 50 52 48 65 50 55
Oddish 45 50 55 75 65 30
Fennekin 40 45 40 62 60 60
Goomy 45 50 35 55 75 40
Vanillite

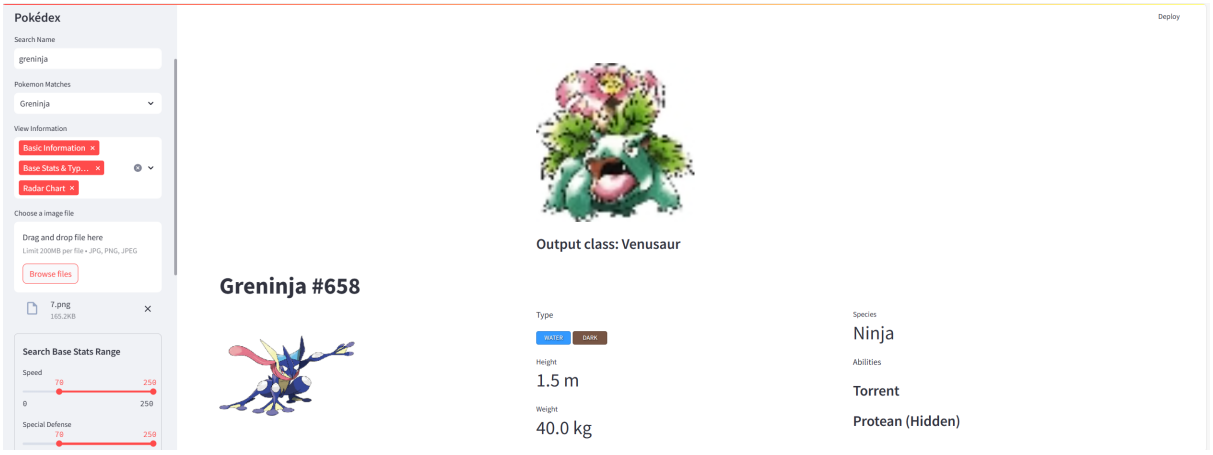

Deploy
Vanillite
Pokedex
Explore
Pokédex
Search Name
Pokemon Matches
Bulbasaur
View Information
Basic Information
Base Stats & Typ...
Radar Chart
Choose a image file
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
Search Base Stats Range
HP
Attack
Defense
Special_attack
Special_defense
Speed

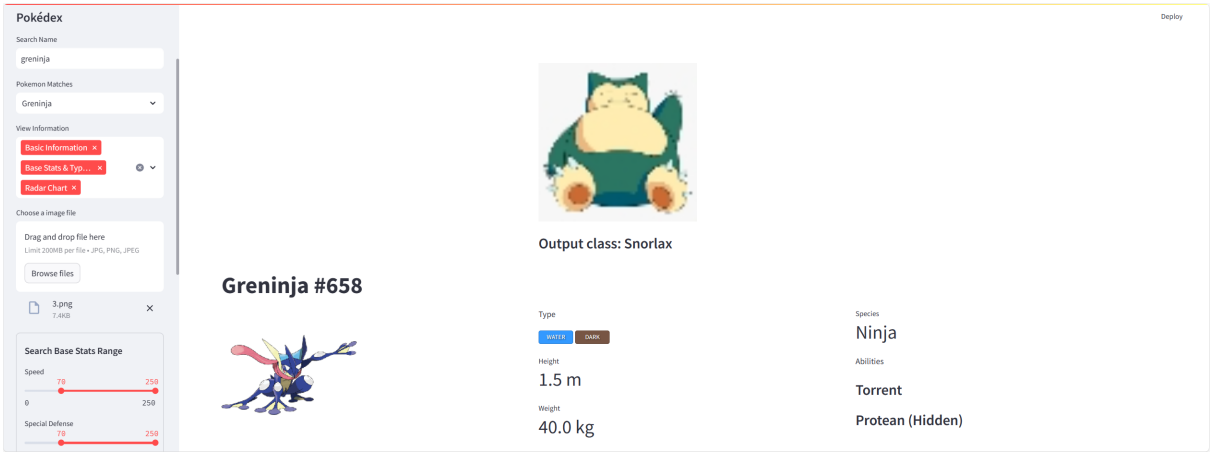


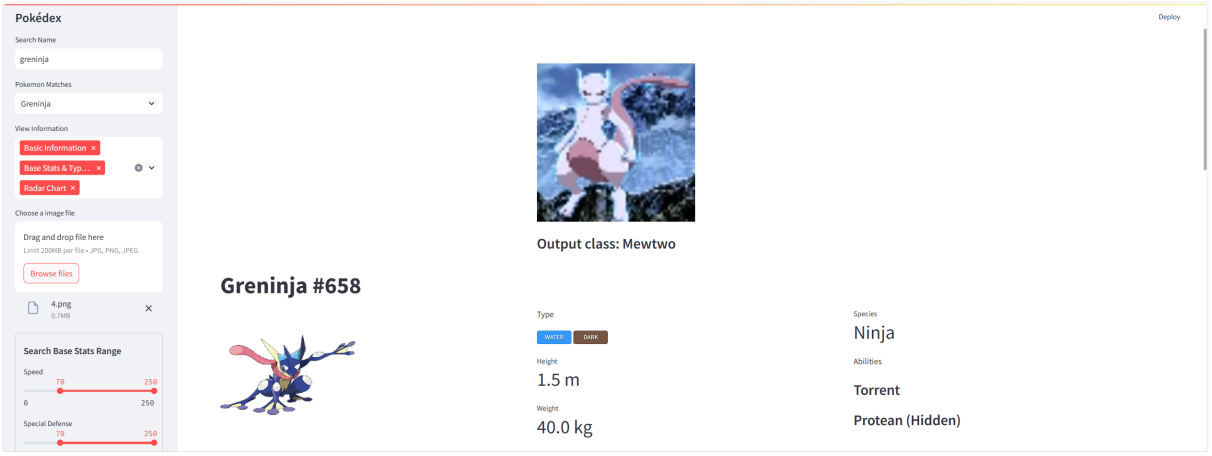


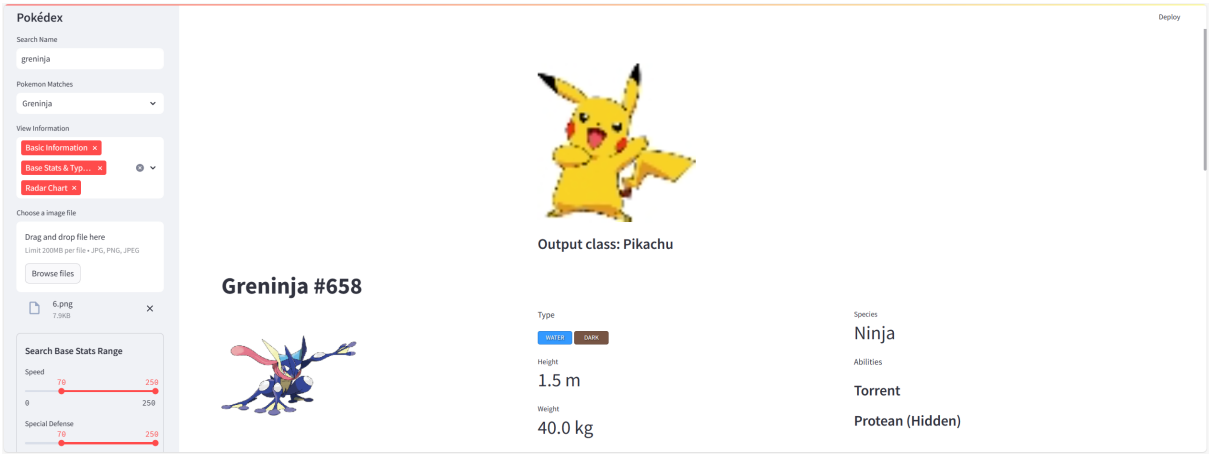
	HP	Attack	Defense	Special_attack	Special_defense	Speed
Mewtwo	106	110	90	154	90	130
Mega Mewtwo Y	106	150	70	194	120	140
Entei	115	115	85	90	75	100
Ho-oh	106	130	90	110	154	90
Rayquaza	105	150	90	150	90	95
Mega Rayquaza	105	180	100	180	100	115
Mega Garchomp	108	170	115	120	95	92
Giratina Altered Forme	150	100	120	100	120	90
Giratina Origin Forme	150	120	100	120	100	90
Arceus	120	120	120	120	120	120
Seismitoad	105	95	75	85	75	74
Kyurem	125	130	90	130	90	95
Black Kyurem	125	170	100	120	90	95
White Kyurem	125	120	90	170	100	95
Xerneas	126	131	95	131	98	99
Yveltal	126	131	95	131	98	99
Solgaleo	137	137	107	113	89	97
Lunala	137	113	89	137	107	97
Nihilego	109	53	47	127	131	103
Eternatus	140	85	95	145	95	130


Pokédex
Search Name
greninja
Pokemon Matches
Greninja
View Information
Basic Information
Base Stats & Typ...
Radar Chart
Choose a image file
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
1.png
133.1KB
Search Base Stats Range
Speed
70
250
0
250
Special Defense
70
250
Deploy
Output class: Charizard
Greninja #658
Type
WATER
DARK
Height
1.5 m
Weight
40.0 kg
Species
Ninja
Abilities
Torrent
Protean (Hidden)


Pokédex
Search Name
greninja
Pokemon Matches
Greninja
View Information
Basic Information
Base Stats & Typ...
Radar Chart
Choose a image file
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
7.png
165.2KB
Search Base Stats Range
Speed
70
250
0
250
Special Defense
70
250
Deploy
Output class: Venusaur
Greninja #658
Type
WATER
DARK
Height
1.5 m
Weight
40.0 kg
Species
Ninja
Abilities
Torrent
Protean (Hidden)


Pokédex
Search Name
greninja
Pokemon Matches
Greninja
View Information
Basic Information
Base Stats & Typ...
Radar Chart
Choose a image file
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
3.png
7.4KB
Search Base Stats Range
Speed
70
250
0
250
Special Defense
70
250
Deploy
Output class: Snorlax
Greninja #658
Type
WATER
DARK
Height
1.5 m
Weight
40.0 kg
Species
Ninja
Abilities
Torrent
Protean (Hidden)


Pokédex
Search Name
greninja
Pokemon Matches
Greninja
View Information
Basic Information
Base Stats & Typ...
Radar Chart
Choose a image file
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
4.png
0.7MB
Search Base Stats Range
Speed
70
250
0
250
Special Defense
70
250
Deploy
Output class: Mewtwo
Greninja #658
Type
WATER
DARK
Height
1.5 m
Weight
40.0 kg
Species
Ninja
Abilities
Torrent
Protean (Hidden)


Pokédex
Search Name
greninja
Pokemon Matches
Greninja
View Information
Basic Information
Base Stats & Typ...
Radar Chart
Choose a image file
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
6.png
7.9KB
Search Base Stats Range
Speed
70
250
0
250
Special Defense
70
250
Deploy
Output class: Pikachu
Greninja #658
Type
WATER
DARK
Height
1.5 m
Weight
40.0 kg
Species
Ninja
Abilities
Torrent
Protean (Hidden)

Scope for improvement

1. Expanding the model’s capability in recognizing all the generations of pokemon
2. Fine tuning and evaluating the model’s accuracy
3. Including additional information such the type weaknesses, moveset calculator, damage calculator, and much more
4. Improving the user interface
5. Introducing an offline mode
6. Allowing community contributions and discussions through chat rooms and forums
7. Extending the model to recognize various merchandise such as pokemon trading cards and plushies with a click of a photo

Bibliography

https://www.pokemon.com/

https://pokemondb.net/

https://bulbapedia.bulbagarden.net

https://medium.com/

https://saturncloud.io/

https://www.tensorflow.org/

https://streamlit.io/