Inteligencja Obliczeniowa - Projekt 1

Alicja Chmura

May 2024

1 Wstęp

W ramach mojego projektu stworzyłam aplikację, która rozpoznaje główne postacie z kultowego sitcomu Friends z lat 90-tych - Rachel, Monicę, Phoebe, Chandlera, Rossa i Joey'ego oraz udostępnia losowe cytaty rozpoznanej postaci. W tym celu wytrenowałam sieć CNN, stworzyłam grafową bazę danych neo4j, API w technologii Flask oraz klienta aplikacji przy użyciu biblioteki React Vite.

2 Zbieranie danych

Znalazłam dataset na platformie Kaggle, który zawierał po 50 zdjęć każdej z postaci. Stwierdziłam, że będzie to dobry punkt wyjścia. W pierwszym kroku stworzyłam prosty skrypt **removedodgyimages.py** mający na celu usunięcie uszkodzonych bądź nieodpowiednich zdjęć, czyli takich, które nie mają rozszerzenia .jpg, .jpeg lub .png oraz takich, z których wczytaniem jest problem.

removedodgyimages.py

friends > ml > removedodgyimages.py > ...

```python
# Data preprocessing - removing dodgy images
import os
import cv2
import imghdr

data_dir = 'data'
image_exts = ['jpeg','jpg', 'png']

for image_class in os.listdir(data_dir):
    for image in os.listdir(os.path.join(data_dir, image_class)):
        image_path = os.path.join(data_dir, image_class, image)
        try:
            img = cv2.imread(image_path)
            tip = imghdr.what(image_path)
            if tip not in image_exts:
                print(f'Image not in ext list {format(image_path)}')
                os.remove(image_path)
        except Exception as e:
            print(f'Issue with image {format(image_path)}')
```

Rysunek 1: Usuwanie nieodpowiednich zdjęć

Po usunięciu kilku uszkodzonych zdjęć ręcznie uzupełniłam dataset. Aby móc sprawnie weryfikować czy klasy są zbalansowane stworzyłam prosty skrypt, który sprawdza ilość zdjęć w każdym z podfolderów folderu data.

```
23 for image_class in os.listdir(data_dir):
24     class_dir = os.path.join(data_dir, image_class)
25     if os.path.isdir(class_dir):
26         num_images = len(os.listdir(class_dir))
27         print(f"{image_class}: {num_images} images")
```

Rysunek 2: Liczenie ilości zdjęć w każdej z klas

3 Pierwszy trening

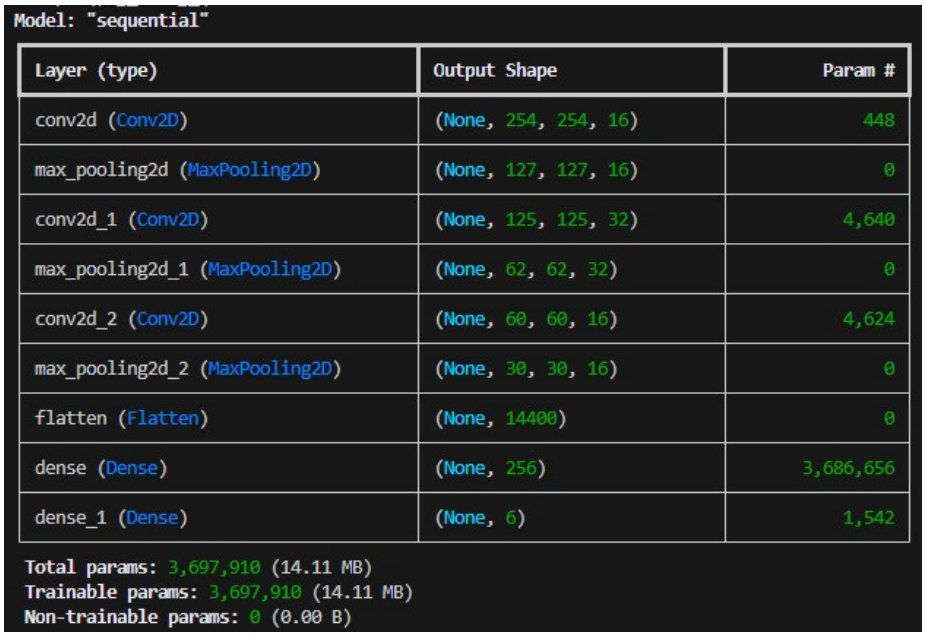
Przyszła pora na stworzenie i wytrenowanie modelu. W tym celu stworzyłam plik **train_model.py**, w którym skorzystałam z bibliotek numpy, matplotlib oraz tensorflow.keras.

W pierwszym kroku przygotowałam dane do treningu wykorzystując prosty preprocessing. Najpierw użyłam metody image_dataset_from_directory żeby załadować zdjęcia i odpowiednie labels. Potem znormalizowałam wartości pikseli i zamieniłam je na numpy iterator.

```
8  data = image_dataset_from_directory('data')
9  data = data.map(lambda x,y: (x/255, y))
10 data.as_numpy_iterator().next()
```

Rysunek 3: Załadowanie danych i preprocessing

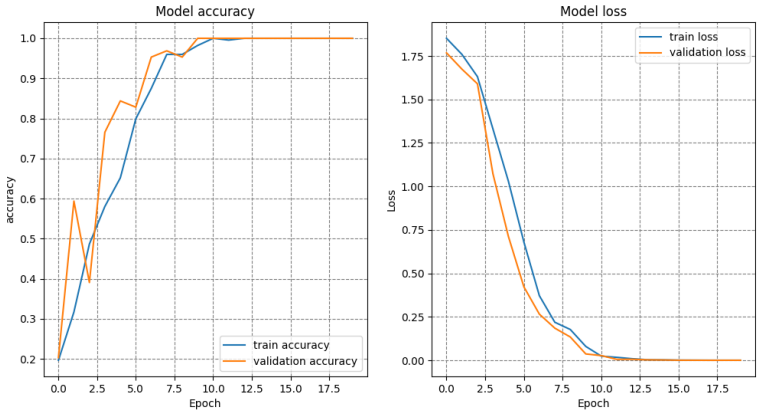
Następnie podzieliłam dane na zbiór treningowy, walidacyjny i testowy w proporcjach 0.7, 0.2 i 0.1. Kolejnym krokiem było stworzenie modelu. Użyłam do tego Sequential API z biblioteki tensorflow.keras. Mój model składa się z 9 warstw. W większości korzystam z funkcji aktywacji relu, a w ostatniej z softmax, dobrze sprawdzającej się w zadaniach z dziedziny categorical classification.



Rysunek 4: Struktura modelu

Przy kompilacji modelu użyłam optimizera adam oraz loss function sparse_categorical_crossentropy, dzięki czemu nie musiałam robić one hot encodingu na labels.

Po kompilacji przyszedł czas na trening. Dodałam callback, który zapisuje model co epokę do pliku .keras, pod warunkiem, że validation accuracy się poprawiła w porównaniu do poprzedniej epoki. Następnie wytrenowałam model przez 20 epok i szybko (już w okolicy 10-11 epoki osiągnęłam 100% train accuracy, validation accuracy i test accuracy.



Rysunek 5: Accuracy i loss pierwszego treningu

4 Przetestowanie modelu

Przyszła pora na przetestowanie modelu na zdjęciach innych niż te, które były w datasecie. W tym celu stworzyłam prosty skrypt, który wczytuje model oraz zdjęcie i przepuszcza je przez model.

Po przetestowaniu modelu na kilku zdjęciach zauważyłam, że jest problem. Mimo że model wytrenował się najlepiej jak mógł, to dokonywał złej klasyfikacji większości zdjęć. Z jakiegoś powodu w większości przypadków rozpoznawał na zdjęciach Rossa, czasami Monicę jeśli na zdjęciu była dziewczyna. Stwierdziłam, że zapewne problem tkwi w datasecie, który nie jest wystarczająco reprezentatywny przez co wytrenowany model słabo radzi sobie z generalizacją na zdjęciach, których nie widział. Postanowiłam, że sama zbiorę więcej zdjęć do datasetu, wykonam upsampling, wytrenuję model o takiej samej strukturze na większej ilości danych i zobaczę czy to poprawi wyniki.

```
friends_model = load_model('friends_50.keras')

img = cv2.imread('rachel.jpg')
plt.imshow(img)
plt.show()

resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()

prediction = friends_model.predict(np.expand_dims(resize/255, 0))[0]
class_labels = ['Chandler', 'Joey', 'Monica', 'Phoebe', 'Rachel', 'Ross']

for i, prob in enumerate(prediction):
    print(f'{class_labels[i]}: {prob:.4f}')
```

Rysunek 6: Przetestowanie modelu

5 Upsampling

Pierwszym miejscem, w które udałam się, aby zwiększyć mój dataset był google. Znalazłam extension, która pobiera grafikę, będącą wynikiem wyszukiwania dla danego promptu. Stwierdziłam jednak, że takie podejście będzie niezbyt wygodne, ponieważ wiele zdjęć nie przedstawia jedynie danej postaci.

Wpadłam na pomysł żeby wykorzystać aplikację Pinterest oraz jej możliwości. Stworzyłam sobie 6 boardów (Chandler, Joey, Monica, Phoebe, Rachel, Ross), po jednej na każdą postać. Już po samej nazwie Pinterest sugerował mi w miarę odpowiednie zdjęcia. Po chwili przypinania pinów, które mi opowiadają sugestie aplikacji były coraz bardziej trafne. Dzięki temu zebranie większego (i zdecydowanie lepszego niż początkowy) datasetu zajęło mi naprawdę mało czasu.

Chandler

509 Pins 1w

Joey

398 Pins 1w

Monica

562 Pins 1w

Phoebe

536 Pins 1w

Rachel

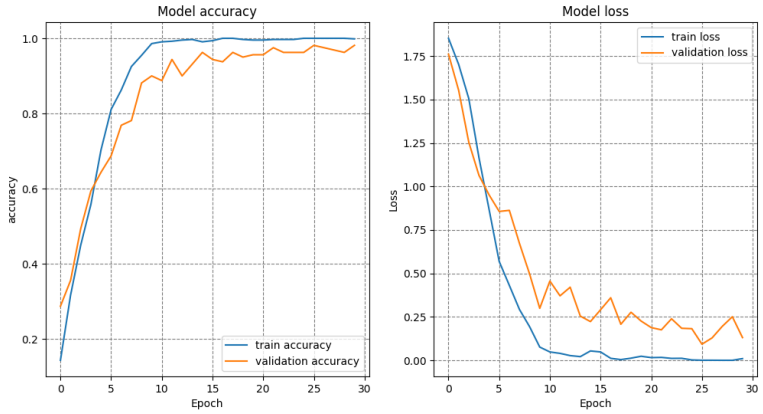
711 Pins 1w

Ross

498 Pins 1w

Kolejnym krokiem było sprawne pobranie pinów z boardów (manulane pobieranie pojedynczych zdjęć zajęłoby wieki). Znalazłam google extension, która pozwala na bulk pobieranie pinów, ale niestety okazała się ona płatna. Poszperałam w internecie i znalazłam darmową stronę, która pozwala na pobieranie 100 pierwszych pinów z danego boardu - na moje potrzeby okazało się to wystarczające.

Pobrałam po 100 pinów z każdego boardu i dodałam je do folderów w moim datasecie (teraz miałam po 150 zdjęć na klasę). Wytrenowałam ten sam model co wcześniej na nowych danych, przez 30 epok i tym razem również udało mi się osiągnąć training accuracy 100%, a validation i test accuracy wyniosły około 98%.



Rysunek 7: Accuracy i loss drugiego treningu

Tym razem po ponownym przetestowaniu na tych samych zdjęciach co wcześniej model dobrze rozpoznawał postacie, co oznacza, że zwiększenie datasetu z 50 do 150 zdjęć na klasę wystarczyło, by rozwiązać wcześniejszy problem z generalizacją. Mając wytrenowany model mogłam przejść do kolejnego etapu projektu.

6 Przetworzenie danych z cytatami

Znalazłam fajny dataset na Kaggle, który zawierał plik csv z cytatami ze wszystkich 10 sezonów serialu podzielonymi na postacie, które je wypowiadają. Jednakże dataset, choć fajny, został uzyskany poprzez webscraping stronki, gdzie ludzie dokonywali transkrypcji odcinków dlatego wymagał trochę preprocessingu zanim mogłam przenieść dane z niego do bazy. Poniżej widać strukturę pliku csv:

	A	B	C	D	E	F
1	author	episode_number	episode_title	quote	quote_ord	season
2	Monica	1	Monica Gets A Roommate	There's nothing to tell! He's just some guy	0	1
3	Joey	1	Monica Gets A Roommate	C'mon, you're going out with the guy! Ther	1	1
4	Chandler	1	Monica Gets A Roommate	All right Joey, be nice. So does he have a h	2	1
5	Phoebe	1	Monica Gets A Roommate	Wait, does he eat chalk?	3	1
6	Phoebe	1	Monica Gets A Roommate	Just, 'cause, I don't want her to go through	4	1
7	Monica	1	Monica Gets A Roommate	Okay, everybody relax. This is not even a c	5	1
8	Chandler	1	Monica Gets A Roommate	Sounds like a date to me.	6	1
9	Chandler	1	Monica Gets A Roommate	Alright, so I'm back in high school, I'm sta	7	1
10	All	1	Monica Gets A Roommate	Oh, yeah. Had that dream.	8	1

Rysunek 8: Dane z cytatami

Na potrzeby preprocessingu stworzyłam plik **preprocess_quotes.py**, w którym posłużyłam się biblioteką pandas. Najpierw upewniłam się, że nigdzie w danych nie ma pustych komórek. Następnie pozbyłam się kolumny quote_order, ponieważ nie była mi ona potrzebna. W kolejnym kroku pozbyłam się cytatów z odcinka "The Two Parts, Part I", ponieważ wkradł tam się błąd i w kolumnie z cytatami zamiast cytatów były imiona postaci. W kolejnym kroku wyfiltrowałam dane tak, aby zachować tylko cytaty wypowiedziane przez 6 głównych postaci, na których klasyfikację się zdecydowałam. Ponieważ ich imiona były zapisywane na różne sposoby, posłużyłam się do tego regexem, jednocześnie ujednolicając zapis imion do nowego arkusza. Kolejnym i najbardziej żmudnym krokiem było usuwanie didaskaliów oraz zastępowanie niedrukowalnych znaków ich drukowalnymi odpowiednikami (w dekodowaniu konkretnych znaków bardzo pomogła mi stronka, z której orginalnie został ściągnięty scenariusz). W ostatnim kroku usunęłam cytaty, które miały mniej niż 11 znaków oraz tam gdzie mogłam dokonałam kapitalizacji liter (np. "I", zamiast "i"). Tym samym skończyłam z arkuszem zawierającym ponad 40 tysięcy rekordów z cytatami.

7 Stworzenie bazy danych

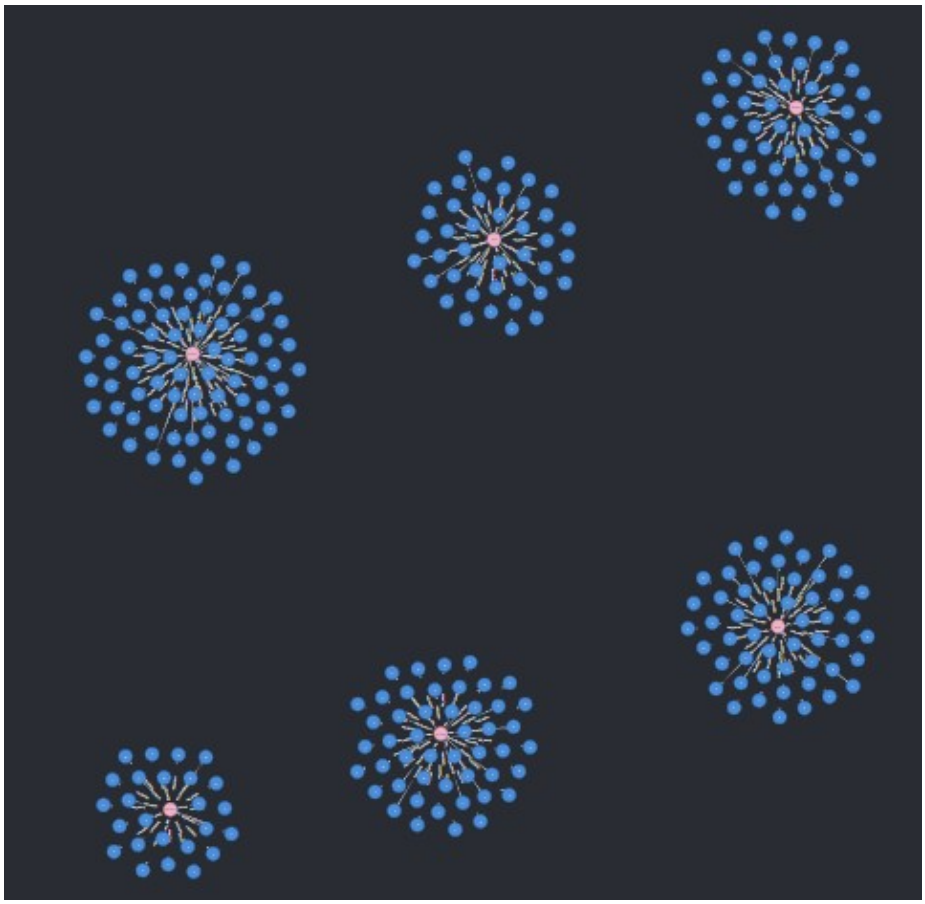
Zdecydowałam się stworzyć bazę grafową neo4j. Początkowo stworzyłam instancję bazy w chmurze przy użyciu Neo4j Aura, ale niestety nie byłam w stanie połączyć się z nią z API, które potem stworzyłam przez błędy z odczytywaniem certyfikatów SSL w Pythonie, dlatego ostatecznie zdecydowałam się stworzyć kontener dockerowy z obrazem bazy Neo4j.

Dodałam do niej 6 node'ów postaci oraz ponad 40 tysięcy node'ów z cytatami (cytaty danej postaci są połączone z nią relacją HAS_QUOTE). Dzięki temu, że dane miałam w pliku csv, zapełniłam bazę przy użyciu prostej query:

```
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/cloudala/Friends-Detection-App/main/db_data/friends_quotes_after_cleanup.csv' AS row
MERGE (author:Author {name: row.author})
WITH author, row
CREATE (quote:Quote {
    episode_number: toInteger(row.episode_number),
    episode_title: row.episode_title,
    quote: row.quote,
    season: toInteger(row.season)
})
CREATE (author)-[:HAS_QUOTE]→(quote)
```

Rysunek 9: Dodanie cytatów do bazy

Fragment dodanych danych (300 węzłów) jest widoczny poniżej:



Rysunek 10: Fragment danych w bazie

Gdy baza była gotowa przyszedł czas na połączenie wszystkiego w całość, czyli na stworzenie API.

8 API we Flasku

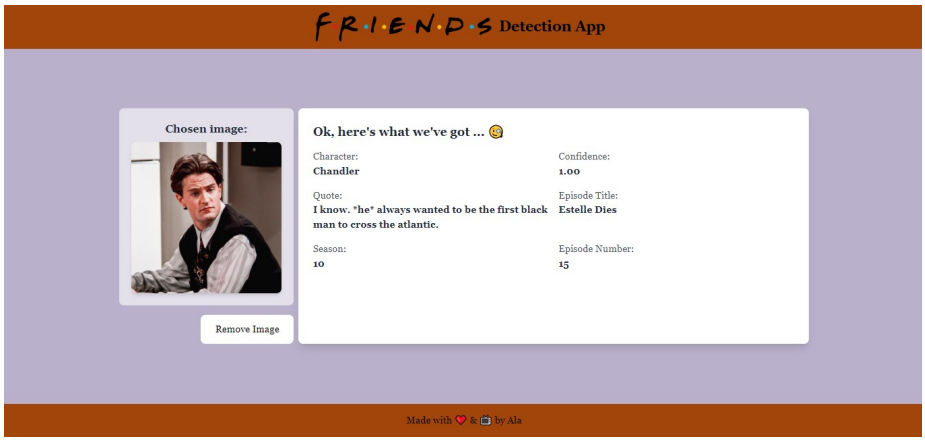
W API najpierw załadowałam wytrenowany model. Następnie przy użyciu dotenv załadowałam zmienne środowiskowe oraz stworzyłam driver bazy danych. Następnie napisałam funkcję, która pobiera losowy cytat (oraz związane z nim informacje), wypowiedziany przez podaną osobę z bazy danych.

W kolejnym kroku stworzyłam /**classify** - główny endpoint aplikacji. Zastanawiałam się jak rozwiązać przesyłanie zdjęć z klienta na serwer tak, aby go nie obciążać i po odrobinie researchu zdecydowałam się na skorzystanie z base64 string. W moim endpoincie przyjmuję go w body pod kluczem image_data, a następnie przy użyciu bibliotek base64, numpy oraz opencv konwertuję na zdjęcie, które mogę przekazać do modelu klasyfikującego. Po przejściu zdjęcia przez model wybieram postać, co do której confidence jest największe i wysyłam zapytanie do bazy o losowy cytat tej osoby. W ostatnim kroku wysyłam informację zwrotną o wykrytej postaci, confidence, z jaką model to zrobił oraz jej cytatem.

9 Aplikacja kliencka w React Vite

Kiedy API było gotowe przyszedł czas na stworzenie klienta. Zdecydowałam się wykonać go w React Vite (jest to technologia najbardziej podobna do już deprecated create react app, jednocześnie dużo szybsza i lżejsza niż np, next.js). Stworzyłam UI pasujące do tematu aplikacji. Do przesyłania zdjęć skorzystałam z react dropzone (komponent umożliwiający drag and drop zdjęć) oraz FileReader API, dzięki któremu mogłam konwertować zdjęcia na base64 stringi i w takiej postaci wysyłać je do API. Po skończonej pracy aplikacja prezentowała się tak:

Rysunek 11: Klient stworzony w React Vite



Rysunek 12: Działanie aplikacji na przykładowym zdjęciu

10 Podsumowanie

Udało mi się z powodzeniem zrealizować cele, które postawiłam sobie we wstępie - stwodzyłam aplikację, która rozpoznajem która postać z serialu Friends jest na zdjęciu i wyświetla dodatkowe informacje na jej temat.

11 Bibliografia

1. https://www.kaggle.com/datasets/amiralikalbasi/images-of-friends-character-for-face-recognition
2. https://www.youtube.com/watch?v=jztwpsIzEGc&pp=ygUbY25uIGhhcHB5IHBlb3BsZSBzYWQgcGVvcGxl
3. https://www.kaggle.com/datasets/ryanstonebraker/friends-transcript
4. https://pl.pinterest.com/
5. Chat GPT