

Aplikacja pozwalająca na pobieranie tekstu ze zdjęć, tłumaczenie i czytanie

Przetwarzanie języka naturalnego

Autorzy

Kamil Sulżycki
Sebastian Stefan
Anna Tomiczek

Kraków, 2024

Spis treści

1. Wstęp i cel projektu.....	3
2. Projekt aplikacji.....	3
2.1. Wymagania funkcjonalne.....	3
2.2. Wymagania niefunkcjonalne.....	4
2.3. Technologie wykorzystane do realizacji projektu.....	4
3. Implementacja projektu.....	5
3.1. Odczytanie tekstu z grafiki.....	6
3.2. Wektoryzacja.....	7
3.3. Tłumaczenie.....	7
3.4. Konwersja tekstu na mowę.....	9
4. Testowanie.....	9
5. Podsumowanie.....	9
Bibliografia.....	10

1. Wstęp i cel projektu

Rozwój technologiczny powinien przyczyniać się do rozwoju społeczeństwa. Tematem, który w ostatnich latach jest przy tej okazji często poruszany jest dostępność, wobec osób niepełnosprawnych lub posiadających inne problemy natury fizycznej i psychicznej. Powstaje więc wiele udogodnień, które powinny wspomagać poruszanie się takich osób wśród nowoczesnych technologii. Udogodnienia te mogą też pomagać w zwykłym, codziennym życiu poprzez pokonywanie barier, które utrudniały do tej pory swobodne funkcjonowanie.

Chcąc przygotować coś, co może być prawdziwie użyteczne, celem niniejszej pracy stało się stworzenie prostej aplikacji, która pozwoli na załadowanie zdjęcia zawierającego np. tekst w języku angielskim, który sprawia jakąś trudność w odczycie przez używającą go osobę. Kolejnym etapem będzie jego głosowe odczytanie go w języku angielskim z pomocą odpowiedniej biblioteki, oraz przygotowanie tłumaczenia na język polski i ponowne użycie biblioteki pozwalającej na czytanie przygotowanego tekstu.

Taka aplikacja, ma szansę pomóc osobom, które mają problem ze wzrokiem, a chciałyby np. odczytać etykietę produktu wydrukowaną zbyt małym rozmiarem czcionki, lub po prostu potrzebują tłumaczenia na język polski jakiegoś tekstu ze zdjęcia lub grafiki znalezionej w Internecie.

Aktualnie na rynku najpopularniejszym narzędziem, które pozwala na odczytywanie tekstów ze zdjęć i ich tłumaczenie na różne języki jest Google Translate. Jest to jednak aplikacja tworzona i ulepszana od wielu lat przez ogromną korporację, która może w nią inwestować ogromne ilości pieniędzy i czasu. Pamiętając jednak poziom tłumaczenia sprzed kilku lat warto zauważyć, że nie zawsze było na tak wysokim poziomie jak teraz.

2. Projekt aplikacji

W niniejszym rozdziale przedstawione zostaną wymagania funkcjonalne oraz нефункционалне dla projektowanej aplikacji, a także wybrane technologie oraz biblioteki wybrane do realizacji projektu.

2.1. Wymagania funkcjonalne

Wymagania funkcjonalne dla niniejszej aplikacji:

- aplikacja powinna umożliwiać użytkownikowi wczytanie pliku;
- aplikacja powinna umożliwiać wybranie języka w którym zostanie odczytany wprowadzony tekst (język polski lub angielski);
- aplikacja powinna wyświetlać wczytany plik;
- aplikacja powinna odczytać tekst z odpowiednią głośnością i szybkością;
- aplikacja powinna wyświetlić czytany tekst w wybranym języku.

2.2. Wymagania niefunkcjonalne

Wymagania niefunkcjonalne dla niniejszej aplikacji:

- aplikacja powinna umożliwiać pracę z plikami w formatach .jpg, .jpeg, .png;
- aplikacja powinna wykonywać odczyt tekstu z wgranych przez użytkownika plików graficznych;
- aplikacja powinna pozwalać na minimalizację istnienia błędów związanych z niepoprawnym odczytaniem tekstu z pliku graficznego;
- aplikacja powinna wykonywać tłumaczenie z języka angielskiego na język polskich tekstów odczytanych z wgranych plików graficznych;
- aplikacja powinna posiadać zaimplementowany słownik pozwalający na tłumaczenie tekstów z języka angielskiego na język polski;
- aplikacja powinna wykonywać lematyzację tekstu w języku angielskim z użyciem odpowiedniej bibliotek, w celu wykonania poprawnego tłumaczenia;
- aplikacja powinna odpowiednio zmieniać głos i akcent skryptu czytającego końcowy tekst w zależności od języka tekstu;
- aplikacja powinna być dostępna dla użytkowników przez co najmniej 99% czasu;
- interfejs aplikacji powinien być intuicyjny i prosty w obsłudze;
- aplikacja powinna być zgodna z najbardziej popularnymi przeglądarkami takimi jak Chrome, Edge, FireFox czy Safari.

2.3. Technologie wykorzystane do realizacji projektu

W toku pracy nad aplikacją ustalono, że będzie ona posiadała architekturę typu klient-serwer opartą o dwa mikroserwisy komunikujące się z sobą za pomocą endpointów restowych. Pierwszy z mikroserwisów będzie odpowiadał za aspekty frontendowe związane z interfejsem użytkownika, a drugi za backend aplikacji, gdzie zostanie zaimplementowana cała logika biznesowa.

Na główną technologię odpowiedzialną za frontend aplikacji wybrano JavaScript w powiązaniu z frameworkiem React. Warstwa wizualna aplikacji ma być prosta i zawierać tylko kilka najważniejszych elementów powiązanych z głównymi funkcjonalnościami aplikacji. Będzie to prosty interfejs aplikacji webowej jednoekranowej, w związku z czym zastosowanie Reacta, który jest bardzo rozbudowanym frameworkiem, wystarczy do stworzenia funkcjonalnego interfejsu użytkownika.

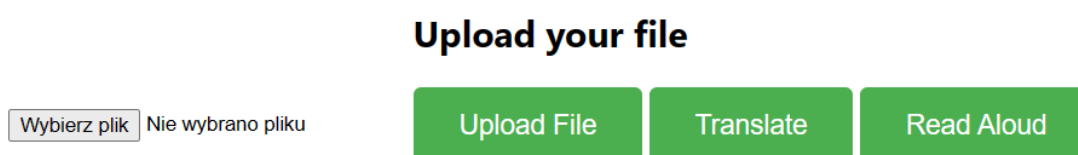
W przypadku technologii backendowych najlepszym wyjściem wydaje się użycie Pythona wraz z frameworkiem Django. W roku pracy i działań biznesowych tworzonej aplikacji, będzie ona zajmowała się w dużej mierze przetwarzaniem danych, a Python w tym aspekcie jest bardzo funkcjonalnym językiem programowania, więc jego wybór jest naturalny. Istnieje wiele bibliotek, które użyte

mogą w znaczący sposób pomóc w implementacji odpowiednich funkcjonalności. Wśród nich są następujące biblioteki:

- pytesseract – narzędzie pozwalające na optyczne rozpoznawanie znaków (OCR). Umożliwia rozpoznawanie i „odczytanie” tekstu osadzonego w grafice lub zdjęciu [1], co umożliwia późniejsze zapisanie odczytanego ciągu znaków do zmiennej;
- pyttsx3 – działające offline narzędzie pozwalające na konwersję zadanego tekstu na mowę [2]. Umożliwia wybranie odpowiedniego głosu i języka w zależności od zainstalowanych w systemie pakietów, a także kontrolę nad głośnością i tempem mowy;
- PIL – biblioteka pozwalająca na przetwarzanie obrazów;
- nltk – platforma umożliwiająca pracę z językiem naturalnym [3]. Umożliwia między innymi wykonanie tokenizacji ciągu znaków;
- sklearn – biblioteka, szeroko wykorzystywana do analizy danych, może zostać wykorzystana do wektoryzacji tekstu na reprezentację numeryczną [4];
- spaCy – biblioteka wspierająca obsługę wielu języków, może być wykorzystywana do wielu transformacji w toku przetwarzania języka naturalnego [5] np. do lematyzacji tekstu na podstawie wczytanego modelu językowego;
- numpy – biblioteka o szerokim zakresie możliwości, w kontekście projektu może być użyta do obliczenia podobieństwa kosinusowego,
- spellChecker - narzędzie pozwalające na korektę przetwarzanego tekstu [6], w niniejszym projekcie może być użyty to poprawienia tekstu odczytanego z grafiki.

3. Implementacja projektu

Wygląd interfejsu niniejszej aplikacji powinien być prosty i pozwalać na intuicyjne jej używanie. W związku z tym został ograniczony do zaimplementowania: przycisków pozwalających na zaimportowanie konkretnej grafiki oraz wybranie języka w którym będzie wykonana konwersja tekstu na mowę, a także pól w których zostanie wyświetlona wgrana grafika oraz tekst w wybranym języku. Interfejs użytkownika przedstawiono na rys. 1 poniżej.



Rys. 1 Wygląd interfejsu aplikacji

3.1. Odczytanie tekstu z grafiki

OCR (ang. Optical Character Recognition) to zestaw technik pozwalających na rozpoznawanie znaków i tekstów w plikach grafik rastrowych. Jest to bardzo skomplikowane zadanie, stosowane między innymi do digitalizacji zasobów bibliotecznych, ułatwiania odczytu pisma odręcznego, a także w zabezpieczeniach stron internetowych CAPTCHA.

Do rozpoznawania znaków wykorzystywane są zaawansowane sieci neuronowe, w związku z czym na potrzeby niniejszego projektu wykorzystano istniejące już narzędzie pytesseract. Jest to biblioteka stworzona na potrzeby Pythona, która wykorzystuje Google Tesseract OCR, czyli już gotową sieć neuronową, która pozwala na rozpoznawanie znaków [7]. Dzięki wykorzystaniu tak zaawansowanego narzędzia, można rozpoznawać znaki występujące w około stu różnych językach, w tym także w języku polskim, który jest bardzo trudny do poprawnej interpretacji przez tego typu narzędzia.

Na listingu 1 przedstawiono metodę `get_text_from_image(file)`, która wykorzystując jedną z metod biblioteki pytesseract (`image_to_string(img)`) odczytuje tekst z grafiki, a następnie zapisuje go w formacie tekstowym do zmiennej, która na późniejszych etapach może zostać wykorzystana do obróbki otrzymanego ciągu znaków. Odczytany tekst może jednak zawierać błędy, w związku z czym warto zweryfikować jego poprawność.

```
def get_text_from_image(file):  
    img = Image.open(file)  
    text = pytesseract.image_to_string(img)  
    return text.replace('\n', ' ')
```

Listing 1. Metoda odczytująca tekst z importowanej grafiki

W tym celu zastosowano narzędzie SpellChecker, które posiada możliwość weryfikacji poprawności ciągów znaków w kilku językach, między innymi w języku angielskim czy francuskim. Jak przedstawiono na listingu 2 utworzono obiekt `spell`, który w kolejnych etapach posłużył do weryfikacji poprawności poszczególnych wyrazów, a w przypadku znalezienia błędu dochodziło do dopasowania najbardziej prawdopodobnego słowa.

```
spell = SpellChecker()  
  
def correct_spelling(self, text):  
    words = text.split()  
    corrected_words = [self.spell.correction(word) for word in words]  
    corrected_text = ' '.join(corrected_words)  
    return corrected_text
```

Listing 2. Metoda poprawiająca poprawność odczytanego tekstu

3.2. Wektoryzacja

W komercyjnych projektach do tłumaczenia tekstu na różne języki najczęściej wykorzystywane jest Google Translate API, co w bardzo prosty sposób, bo już poprzez zaimportowanie jednej biblioteki `googletrans`, pozwala na dostęp do wysokiej jakości tłumaczeń, pomiędzy wieloma językami z całego świata. W tym projekcie ta biblioteka nie została jednak wykorzystana, co pozwoliło na samodzielne stworzenie prostego tłumacza, który w aktualnej formie pozwala na przetłumaczenie prostych tekstów z języka angielskiego na język polski.

Stworzona samodzielnie biblioteka, która zawiera tłumaczenia dostępna jest w osobnym pliku `.txt` importowanym na początku działania programu, a jej zawartość zapisywana jest z zmiennej będącej słownikiem, która w późniejszym czasie będzie wykorzystana podczas procesu tłumaczenia. Słownik zawiera słowa lub zwroty w formie bezokoliczników. Poniżej na rys. 2 przedstawiono fragment słownika.

```
9      make:robic
10     us:nas
11     lose:stracic
12     interest:zainteresowanie
13     in:na
14     work:praca
15     and:i
```

Rys. 2. Fragment słownika angielsko-polskiego

W następnym etapie pracy wykonywana jest wektoryzacja wartości słownikowych z wykorzystaniem obiektu `vectorizer` będącego instancją `TfidfVectorizer` z biblioteki `sklearn`. Metoda `fit_transform()` wykorzystuje słowa-klucze z wcześniej zaimportowanej biblioteki w celu stworzenia macierzy TF-IDF. W następnym kroku uzyskana macierz jest konwertowana w celu prostszego dalszego przetwarzania. Opisane etapy przedstawiono na listingu 3.

```
def vectorize():
    X = vectorizer.fit_transform(list(dictionary_en_pl.keys()))
    return np.array(X.todense().copy())
```

Listing 3. Metoda wykonująca wektoryzację

3.3. Tłumaczenie

Kolejnym etapem jest wykonanie tokenizacji i lematyzacji tłumaczonego tekstu z pomocą wcześniej utworzonego obiektu `nlp` do którego załadowany został odpowiedni dla języka angielskiego model językowa naturalnego – `en_core_web_sm`. W wyniku działania na danych zwrócone zostają słowa-tokeny

oraz ich atrybuty, takie jak części mowy. Kolejnym krokiem jest utworzenie listy która zawiera poszczególne lematy jak pokazano na listingu 4, który przedstawia także przesyłanie poszczególnych tokenów do metody tłumaczącej poszczególne słowa oraz przesłanie ostatecznie przetłumaczonego tekstu z jednoczesnym usunięciem znaków interpunkcyjnych.

Tokenizacja jest to podział tekstu na mniejsze jednostki nazywane tokenami. Ma ona na celu ułatwienie przetwarzania uzyskanych danych. Lematyzacja to natomiast przekształcenie słów to ich podstawowych form, w przypadku lematyzacji warto zwrócić uwagę, na zagrożenie utratą zbyt wielu informacji.

```
def translate_text(text):
    doc = nlp(text)
    lemmatized_words = [token.lemma_ for token in doc]
    translated_text = ' '.join(translate_word(word) for word in lemmatized_words)
    return translated_text.replace(' ,', ',').replace(' .', '.')
```

Listing 4. Fragment metody wykonującej tokenizację i lematyzację

W metodzie `translate_word(word)` wykonywana jest zmiana znaków na małe litery, co ma pomóc w poprawnym tłumaczeniu z użyciem prostego słownika. Następnie z pomocą zmiennej `highest_similarity` określane jest początkowe prawdopodobieństwo, równe -1, że analizowany token jest odpowiednikiem słowa ze słownika. Jest to związane z obliczaniem podobieństwa kosinusowego, którego wynik waha się pomiędzy -1, a 1 i im jest wyższy, tym wyższe jest podobieństwo. Zmienna `best_match` inicjalizowana jest początkowo tłumaczonym słowem. W przypadku znalezienia wyższego od aktualnie zapisanego prawdopodobieństwa do zmiennej będzie przypisywane najlepsze tłumaczenie. Natomiast w przypadku nie znalezienia żadnego podobnego tłumaczenia słowo pozostanie w tekście końcowym nieprzetłumaczone. Cały ten proces opisuje metoda przedstawiona na listingu 5.

```
def translate_word(word):
    translated_word = word.lower()
    highest_similarity = -1
    best_match = translated_word

    for candidate, candidate_vector in zip(dictionary_en_pl.keys(), word_vectors):
        similarity = calculate_similarity(word.lower(), candidate_vector)
        if similarity > highest_similarity:
            highest_similarity = similarity
            best_match = dictionary_en_pl[candidate]

    return best_match
```

Listing 5. Metoda tłumacząca poszczególne słowa

Samo wyliczanie podobieństwa kosinusowego jest także zamknięte w osobnej metodzie o nazwie `calculate_similarity(word, vector)`, gdzie do obliczeń wykorzystany jest odpowiedni wzór.

Efektem ciągu tych działań będzie przetłumaczony słowo po słowie tekst w języku polskim.

3.4. Konwersja tekstu na mowę

Ostatnim etapem działania aplikacji jest wykonanie konwersji otrzymanego tekstu na mowę. W związku z tym, że w aplikacji przewidziana jest możliwość czytania w różnych językach do funkcji `text_to_speech(text, language)`, oprócz tekstu do odczytania, przekazywana jest też wartość określająca ID języka. Używana tutaj biblioteka `pyttsx3` umożliwia określenie zarówno szybkości czytania i głośności, ale także języka i typu generowanego głosu. Dostępne do wyboru są języki, których pakiety zostały zainstalowane na używanym sprzęcie. W tym przypadku przekazując do metody wartość 0 poprawnie zostanie odtworzony tekst w języku polskim, a w przypadku wartości 1 w języku angielskim. Przebieg działania metody przedstawiono na listingu 6.

```
def text_to_speech(text, language):
    engine = pyttsx3.init()
    engine.setProperty('rate', 150)
    engine.setProperty('volume', 0.9)
    voices = engine.getProperty('voices')
    engine.setProperty("voice", voices[language].id)
    engine.say(text)
    engine.runAndWait()
```

Listing 6. Metoda pozwalająca na konwersję tekstu na mowę

4. Testowanie

W ramach przetestowania aplikacji sprawdzone zostało poprawne działanie interfejsu użytkownika z jego wszystkimi funkcjonalnościami. Zweryfikowano także poprawność działania funkcjonalności na podstawie kilku przygotowanych grafik zawierających różne teksty. Testy te pozwoliły wyeliminować niektóre błędy powstające na etapie wykonywania skryptu odczytującego tekst z grafik.

5. Podsumowanie

Wykonana w trakcie niniejszego projektu aplikacja pozwoliła na skuteczne zaimplementowanie wszystkich zaplanowanych funkcjonalności, a jej działanie można uznać za prawidłowe. Na podstawie wykonanego projektu można zauważyć,

jak trudna jest praca z językiem naturalnym nawet dla dużych i rozbudowanych narzędzi, a wydaje się ona jeszcze bardziej wymagająca, jeśli zamiast języka angielskiego trzeba użyć tak rozbudowanego fleksyjnie i składniowo języka jak polski.

Trzeba przyznać, że jakość zaimplementowanego tłumaczenia nie jest zbyt dobra. Jednak jego poprawienie wymagałoby bardzo dużo dodatkowej pracy. Pod tym względem język polski jest jednym z bardziej skomplikowanych języków. Jest to jednak jedna z tych rzeczy, które należałoby usprawnić w przypadku próby dalszego rozwoju aplikacji. Inną rzeczą, którą można dodać jest obsługa kolejnych języków, zarówno podczas tłumaczeń, jak i przekształcania tekstu na mowę. Dodatkową funkcjonalnością takiej aplikacji może być też tworzenie krótkich podsumowań odczytanych tekstów, a nie koniecznie skupianie się tylko na oryginalnej ich wersji.

Możliwości jest dużo, co pozwala sądzić, że mimo iż program już teraz jest funkcjonalny, wciąż może być dużo lepszy. Widać, że w branży IT w aspekcie pracy nad przetwarzaniem języka naturalnego wciąż jest dużo do zrobienia i ulepszenia na każdym poziomie.

Bibliografia

1. <https://github.com/h/pytesseract> [01.05.2024]
2. <https://github.com/nateshmbhat/pyttsx3> [01.05.2024]
3. <https://www.nltk.org/> [01.05.2024]
4. <https://scikit-learn.org/stable/> [01.05.2024]
5. <https://spacy.io/> [01.05.2024]
6. <https://pypi.org/project/pyspellchecker/> [21.05.2024]
7. <https://github.com/tesseract-ocr/tesseract> [02.05.2024]