AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Inżynieria Oprogramowania 2020

Temat projektu:
Rozpoznawanie znaków języka migowego

Dokumentacja deweloperska

Kryptonim projektu: Sign

Lider:
Przemysław Jabłecki

Członkowie zespołu:
Anna Świadek
Grzegorz Sroka
Filip Ślazyk

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Informatyki, Elektroniki i Telekomunikacji
Kierunek Informatyka

1. Zakres realizacji

1.1. Funkcjonalności

W ramach projektu “Sign” zaimplementowaliśmy aplikację pozwalającą na naukę znaków języka migowego, należących do alfabetu amerykańskiego. Aplikacja wyświetla znak, jaki użytkownik powinien pokazać do kamery. Jednocześnie w dedykowanym oknie wyświetlany jest aktualnie rejestrowany obraz z kamery oraz, w osobnym oknie, obraz podlegający właściwej klasyfikacji, wstępnie przetworzony w celu ekstrakcji cech ułatwiających detekcję.

Aplikacja, poza główną funkcjonalnością rozpoznawania znaku, pozwala na wyświetlenie statystyk dotyczących nauki. Dostępne są pełne dane zawierające wpisy dla pojedynczego wyświetlenia litery. Ponadto można wyświetlić też łączne statystyki zebrane dla poszczególnych liter.

Nauka znaków przez użytkownika jest dodatkowo wspierana poprzez aktywną selekcję liter losowanych do wyświetlenia w trakcie treningu. Pokazywane są tylko te litery, które nie zostały pokazane co najmniej 10 razy i przy których łączny stosunek sukcesów względem wszystkich prób nie przekracza 70%.

1.2 Struktura programu

Program składa się z kilku zasadniczych modułów:

- moduł backendu, składający się z podmodułów:
    - serwera,
    - bazy danych,
    - modułu rozpoznawania znaków (dokonującego także preprocessingu obrazu uzyskanego z kamery).
- moduł frontendu.

Sposób implementacji i działania poszczególnych modułów został opisany bardziej szczegółowo w dalszych rozdziałach.

2. Wymagane biblioteki

Wszystkie wymagane biblioteki do uruchomienia programu znajdują się w pliku Requirements.txt. Na ten moment skorzystaliśmy z bibliotek:

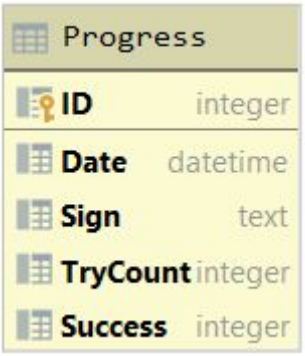
- flask 1.1.2 - serwer flask jest użyty do połączenia modułu serwera z modelem front-endu
- keras 2.3.1 - wykorzystywany w module rozpoznawania obrazu

- opencv-python 4.2.0.34 - używamy go w module rozpoznawania obrazu
- numpy 1.18.4 - standardowa biblioteka służąca do obliczeń matematycznych w języku python mająca różnorakie zastosowania, w programie wykorzystywana jest tworzenia klasyfikatora oraz rozpoznawania obrazów

3. Baza danych

Baza danych opiera się na technologii SQLite (z pomocą wbudowanej biblioteki Pythona sqlite3). Baza zapisywana jest do pliku sign.db.

Przy uruchamianiu programu tworzona jest tabela Progress.



Zawiera ona informacje o sesjach treningowych i wyniku danej sesji. Posiada następujące kolumny:

- ID (Klucz główny) - kolumna służąca do identyfikacji wpisu
- Date - data i godzina zakończenia danej sesji
- Sign - znak (litera), która była trenowana w trakcie danej sesji
- TryCount - ilość nieudanych prób pokazania powyższego znaku, liczona do momentu dobrego pokazania znaku lub naciśnięcia przycisku “skip” na front-endzie
- Success - czy sesja treningowa się powiodła, 1 - gdy użytkownikowi udało się pokazać dobry znak, 0 - jeśli nacisnął “skip”

Do łączenia się z bazą została wykorzystana funkcja with_connection, która wykorzystywana jest później jako dekorator. Funkcja ta zwraca połączenie bazy.

```
def with_connection(func):
    def with_connection_(*args, **kwargs):
        database_path = 'sign.db'
        connection = None
        res = None
        try:
            connection = sqlite3.connect(database_path)
            res = func(connection, *args, **kwargs)
        except sqlite3.DatabaseError as e:
            print(e)
        finally:
```

```
            if connection:
                connection.close()
        return res

    return with_connection_
```

Funkcje get_progress_table oraz get_stats_progress_table są odpowiedzialne za pobieranie danych z bazy. Funkcja get_stats_progress_table zwraca informacje o wszystkich sukcesach i porażkach w pozywaniu danej litery. Informacje te są następnie wykorzystywane w pokazywaniu statystyk uczenia się.

```
@with_connection
def get_progress_table(connection):
    c = connection.cursor()
    c.execute("SELECT * FROM Progress ORDER BY Date DESC LIMIT 100")
    return c.fetchall()


@with_connection
def get_stats_progress_table(connection):
    c = connection.cursor()
    query = '''select Sign,
      (select count(*)
       from Progress as p1
       where p.Sign =p1.Sign and p1.Success=1) as Successes,
      (select count(*)
       from Progress as p2
       where p.Sign =p2.Sign and p2.Success=0) as Failures
    from Progress as p
    group by Sign;'''
    c.execute(query)
    return c.fetchall()
```

Funkcje get_data oraz get_stats_date zwracają listę słowników, które następnie będą przesyłane do aplikacji front-endowej. Taki format danych jest potrzebny do odpowiedniej serializacji wiadomości.

```
def get_data():
    data = get_progress_table()
    result = []
    for item in data:
        (id, date, sign, count, success) = item
        d = {
            'id': id,
            'date': date,
            'sign': sign,
            'count': count,
            'success': success
        }
```

```
        result.append(d)
    return result


def get_stats_data():
    data = get_stats_progress_table()
    result = []
    for item in data:
        (sign, successes,failures) = item
        d = {
            'sign': sign,
            'successes': successes,
            'failures': failures
        }
        result.append(d)
    return result
```

4. Serwer

Do zaimplementowania backendu naszej aplikacji użyliśmy serwera Flask. Implementacja tego modułu polegała na definiowaniu akcji, jakie podejmował serwer w zależności od zapytania uzyskanego z modułu front-endu.

Metoda służąca do cyklicznego sprawdzania, czy litera została dobrze rozpoznana.

```
@app.route('/check', methods=['GET'])
@cross_origin()
def check():
    global ATTEMPTS_COUNT
    last_letter = rs.last_letter()
    if LETTER_TO_BE_SHOWN == last_letter:
        insert_progress([datetime.today(), LETTER_TO_BE_SHOWN, ATTEMPTS_COUNT,
1])
        ATTEMPTS_COUNT = 0
        set_random_letter()
        return jsonify({"success": True, "last_letter": last_letter,
"new_letter": LETTER_TO_BE_SHOWN})
    else:
        ATTEMPTS_COUNT += 1
        return jsonify({"success": False, "last_letter": last_letter,
"new_letter": None})
```

Metoda wykorzystywana przez front-end do wyboru litery wyświetlanej na nim aktualnie litery.

```
@app.route('/currentletter', methods=['GET'])
@cross_origin()
def get_stats():
    return jsonify({"letter": LETTER_TO_BE_SHOWN})
```

Metoda, która jest wykorzystywana w momencie kliknięcia na przycisk “Skip” na front-endzie.

```
@app.route('/skip', methods=['POST'])
@cross_origin()
def skip():
    global ATTEMPTS_COUNT
    last_letter = rs.last_letter()
    insert_progress([datetime.today(), LETTER_TO_BE_SHOWN, ATTEMPTS_COUNT, 0])
    ATTEMPTS_COUNT = 0
    set_random_letter()
    return jsonify({"success": False, "last_letter": last_letter, "new_letter":
LETTER_TO_BE_SHOWN})
```

Metoda służąca do pobierania z Bazy Danych, danych do zakładki Sessions

```
@app.route('/sessions', methods=['GET'])
@cross_origin()
def get_letter():
    data = get_data()
    return jsonify({"data": data})
```

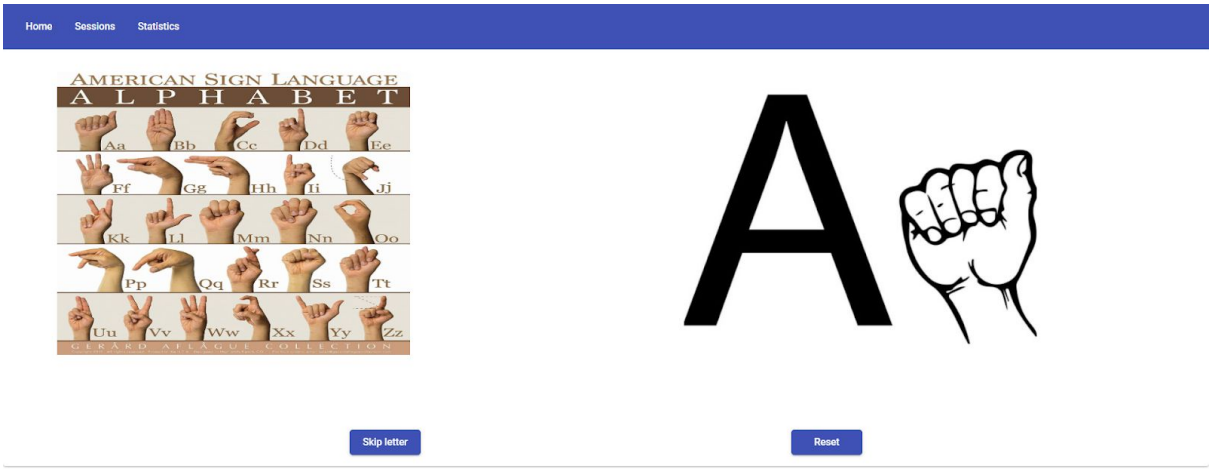
Metoda służąca do pobierania z Bazy Danych, danych do zakładki Statistics

```
@app.route('/stats', methods=['GET'])
@cross_origin()
def get_stats_things():
    data = get_stats_data()
    return jsonify({"data": data})
```

Metoda służąca do resetowania statystyk w bazie danych w celu rozpoczęcia treningu od nowa.

```
@app.route('/reset', methods=['POST'])
@cross_origin()
def reset_stats():
    delete_stats_data()
    return jsonify({})
```

5. Moduł front-endu

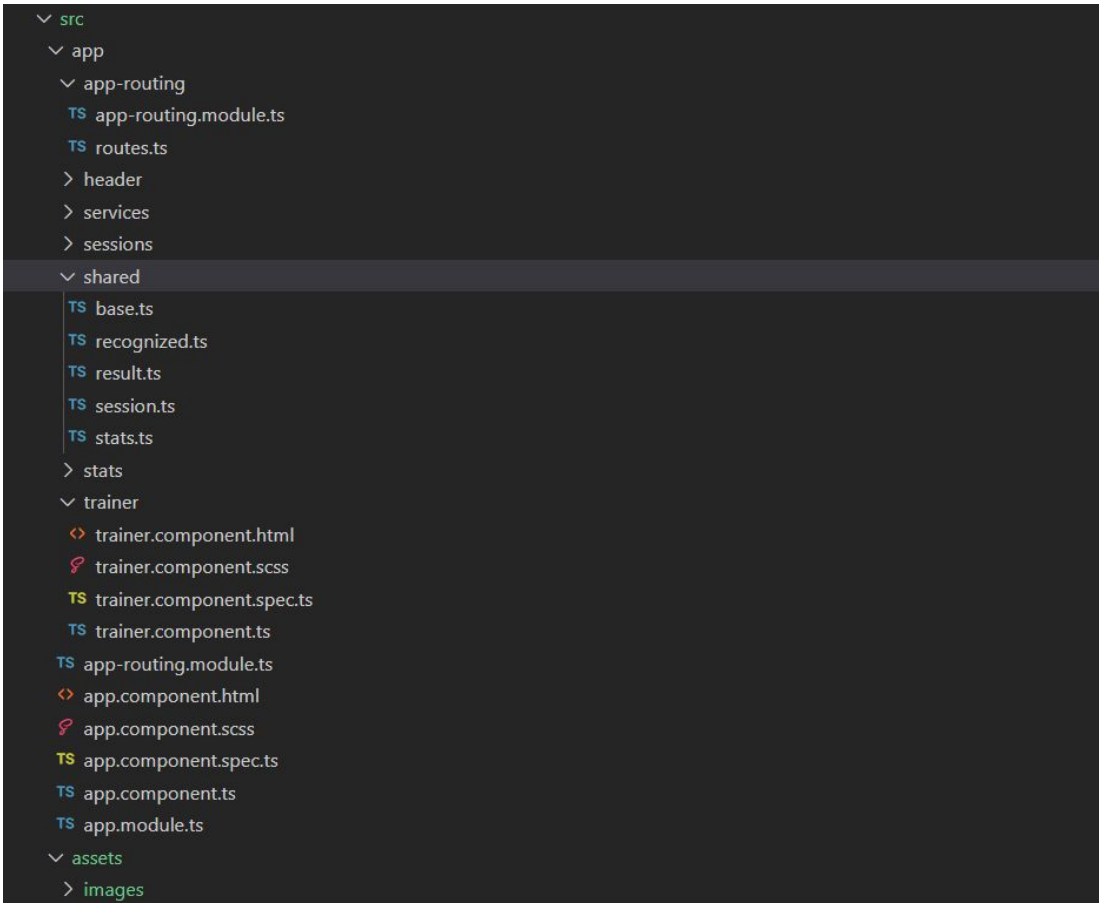


Strona główna projektu "Sign".

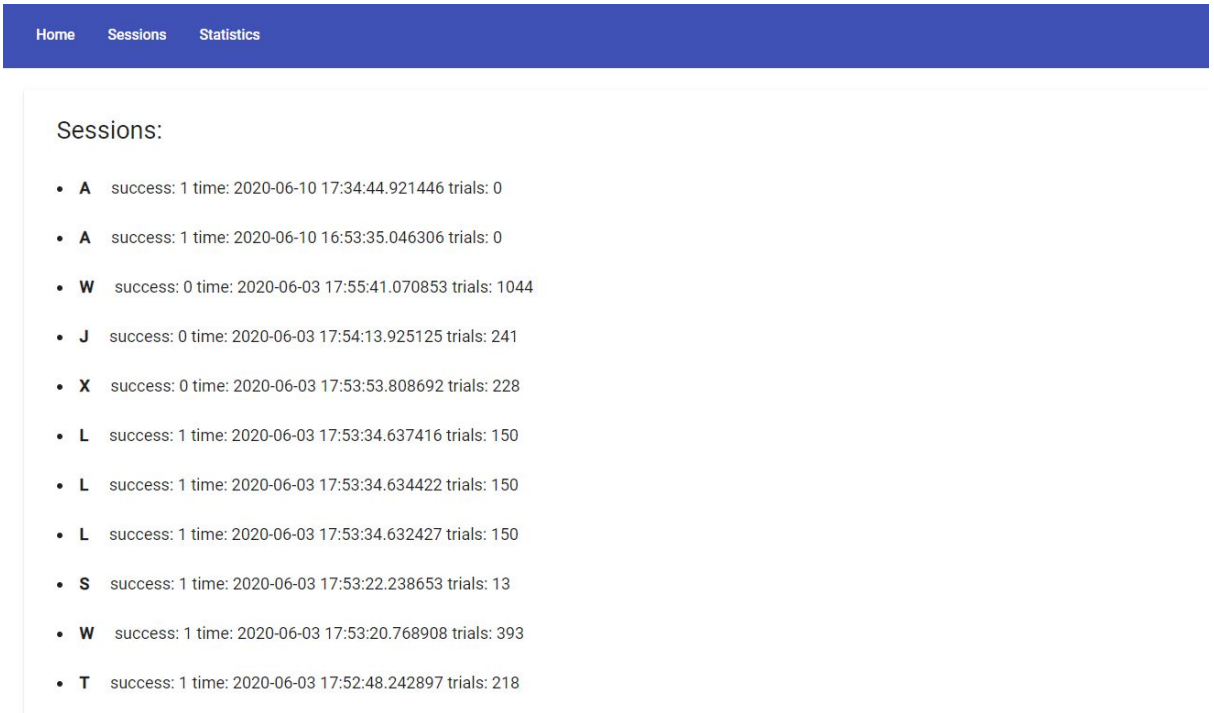
Moduł front-endu został zaimplementowany w języku TypeScript przy pomocy frameworka Angular, HTML-a, SCSS oraz menagera pakietów - npm. Projekt jest podzielony na komponenty, serwisy oraz moduły.
Warto wymienić tutaj m.in:

- komponent "*trainer*", który jest odpowiedzialny za logikę strony głównej aplikacji, w zakres jego funkcjonalności wchodzi sterowanie przełączaniem aktualnie oczekiwanej litery języka migowego, resetowanie sesji,
- komponent "*stats*", odpowiedzialny za wyświetlanie statystyk obliczonych na podstawie sesji użytkownika,
- komponent "*sessions*" odpowiada za wypisywanie rezultatów użytkownika z poprzednich sesji,
- komponent "*header*" służy do wyświetlania opcji w nagłówki i obsługiwanie routingu pomiędzy kartami
- serwis "*letter*" komunikuje się z serwerem, w celu ustalenia jaka litera powinna być aktualnie wyświetlana użytkownikowi
- serwis "*stats*" pobiera aktualne statystyki użytkownika
- serwis "*sessions*" pobiera oraz resetuje historię sesji użytkownika.
- moduł routing odpowiada za routing pomiędzy kartami aplikacji.

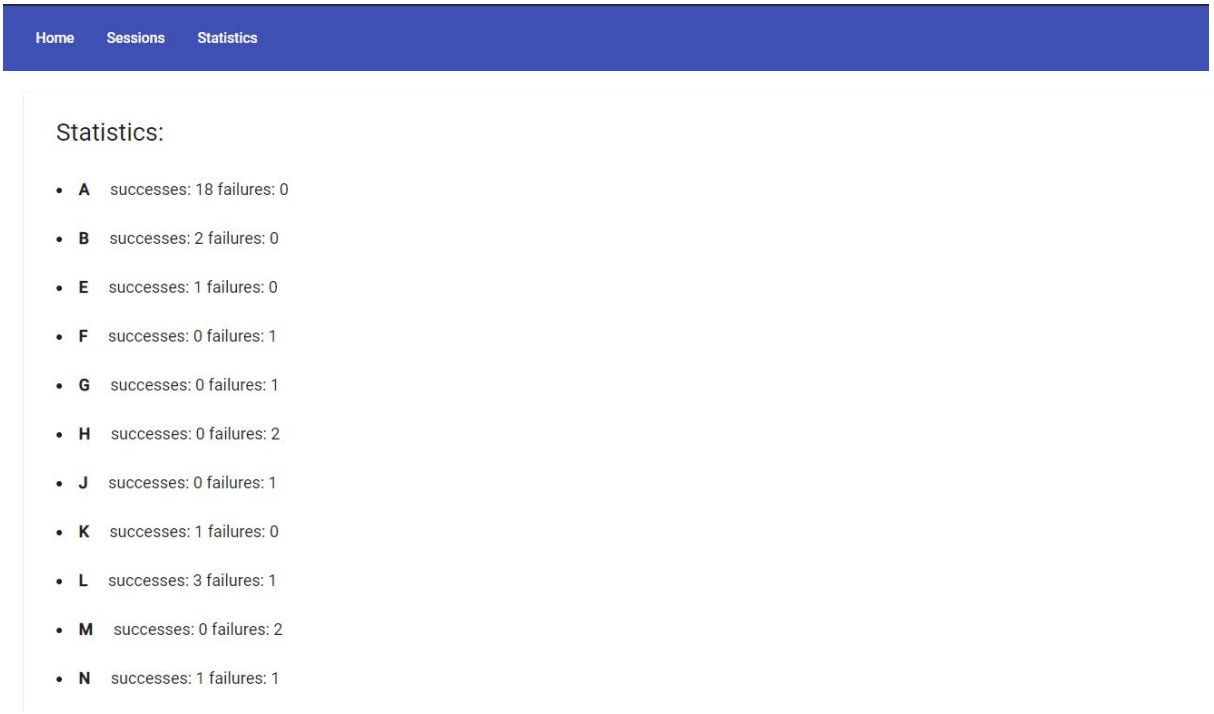
Wszystkie zależności części front-endowej aplikacji znajdują się w pliku package.json, w celu zainstalowania ich na bieżącym serwerze, należy wykonać komendę "`npm install`".



Struktura projektu



Przegląd sesji użytkownika



Przegląd statystyk użytkownika

6. Moduł rozpoznawania obrazu

6.1. Kamera

Do przechwytywania obrazu z kamery wykorzystywana jest biblioteka OpenCV (cv2 w Pythonie).

Funkcja zwracająca obiekt kamery:

```
def create_cam_obj():
    cam = cv2.VideoCapture(1)
    if cam.read()[0] == False:
        cam = cv2.VideoCapture(0)
    return cam
```

Funkcja show odpowiada za wyświetlanie obrazu z kamery. Wyświetlane są dwa okienka, jedno pokazuje rzeczywisty obraz z kamery, drugie przetworzony obraz z zarysowanymi krawędziami.

```
def show(h, img, text, thresh, w, x, y):
    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 255), 2)
    cv2.imshow("Recognizing gesture", img)
    cv2.imshow("thresh", thresh)
```

6.2. Rozpoznawanie obrazu

Model został wytrenowany przy pomocy frameworka Tensorflow za pomocą konwolucyjnej sieci neuronowej. Wytrenowany model został zapisany do pliku HDF5 (hierarchical data format), aby aplikacja nie musiała przechodzić przez długotrwały okres trenowania. Plik ze zrzutem modelu jest dekompresowany przy pomocy biblioteki z pakietu Tensorflow - Keras. Gdy moduł odpowiedzialny za wczytywanie obrazu z kamery przekaże go do modułu rozpoznawania obrazu, klatka zostanie przetworzona, przeniesiona do skali szarości i przekazana predykcji biblioteki Keras.

```
def keras_predict(model, image):
    processed = keras_process_image(image)
    pred_probab = model.predict(processed)[0]
    pred_class = list(pred_probab).index(max(pred_probab))
    return max(pred_probab), pred_class
```

Maksymalne prawdopodobieństwo reprezentujące przynależność obiektu do klasy zostaje odwzorowane w rozpoznany znak języka migowego.

7. Możliwości dalszej rozbudowy i ulepszania działania programu

Na ten moment nasz program działa na zasadzie rozpoznawania pojedynczych znaków.
Przewidujemy możliwość rozbudowy działania programu tak, żeby można było go wykorzystać do nauki słów z języka migowego. W tym celu należy do projektu dodać moduł rozpoznawania słów.

Problemem, z którym deweloperzy będą musieli się zmierzyć będzie pytanie, jak rozdzielić od siebie kolejne słowa. Po konsultacji z Prowadzącym sugerujemy jedno z następujących rozwiązań. Rozpoczęcie nowego słowa sygnalizujemy:

- naciśnięciem klawisza na klawiaturze.
- przybliżeniem dłoni pokazującej znak do kamery.
- odczekaniem pewnego odstępu czasowego

W celu łączenia znaków w słowa sugerujemy użycie biblioteki Natural Language Toolkit[1].

[1] https://www.nltk.org/