FACIAL RECOGNITION USED IN REST WEB APPLICATION

***Abstract*— Aceasta lucrare va prezenta multiplele aplicații ale recunoașterii faciale în viața de zi cu zi. Ne propunem pana la final sa explicăm cum am putea face un sistem de autentificare bazat pe recunoaștere faciala. De asemenea în prezentul articol se va explica implementarea unei platforme în care profesorii se vor putea autentifica pentru a avea acces la notele elevilor precum și rapoarte extrase din acele note și prezente.**

***Keywords*— Recunoaștere faciala, Python, JavaScript, React, Flusk, REST**

Introducere

Un sistem de recunoaștere facială este o tehnologie capabilă să potrivească o față umană dintr-o imagine cu o bază de date de fețe, utilizată de obicei pentru autentificarea utilizatorilor prin serviciile de verificare a identității, funcționează prin identificarea și măsurarea trăsăturilor faciale dintr-o anumită imagine.

Recunoașterea facială automată a fost inițiată în anii 1960. Woody Bledsoe, Helen Chan Wolf și Charles Bisson au lucrat la utilizarea computerului pentru a recunoaște fețele umane. Proiectul lor timpuriu de recunoaștere facială a fost supranumit „om-mașină”, deoarece coordonatele trăsăturilor faciale dintr-o fotografie trebuiau stabilite de un om înainte ca acestea să poată fi utilizate de computer pentru recunoaștere [1].

Dorim eficientizarea procesului academic, și din acest motiv ne-am propus să creăm o aplicație în care profesorii se pot înregistra pe baza unei imagini. În acest site profesorii vor putea sa isi vada studenții împreuna cu prezentele/absențele acestora de asemenea vor fi prezentate și statistici create pe baza notelor elevilor, precum și prezente/absente.

Pentru a implementa aceasta aplicatie am avut nevoie de mai multe framework-uri. Pentru partea de client am folosit mediul de dezvoltare React. React (cunoscut și sub numele de React.js sau ReactJS) este o bibliotecă JavaScript front-end open-source [2] pentru construirea de interfețe utilizator sau componente UI.

Pe langa partea client vom avea nevoie și de un server care v-a procesa toate datele primite de la aplicația front-end. Pentru crearea acestui API ne vom folosi de Django [3]. Este un web framework scris în Python. În interiorul acestui API scris în Python vom avea și partea de procesare de imagini și anume recunoașterea facială. Pentru a avea parte de acest feature in aplicatia noastra am utlizat libraria open source OpenCV [4].

Recunoaștere Faciala

S-ar putea să fii bun la recunoașterea fețelor. Probabil că vi se pare foarte ușor să identificați chipul unui membru al familiei, al unui prieten sau al unei cunoștințe. Cunoașteți trăsăturile feței - ochii, nasul, gura - și cum se reunesc.

Așa funcționează un sistem de recunoaștere facială, dar la o scară mai largă, algoritmică. Acolo unde vedeți o față, tehnologia de recunoaștere vede datele. Aceste date pot fi stocate și accesate.

Deci, cum funcționează recunoașterea facială? Tehnologiile variază, dar iată pașii de bază:

Pasul 1. O fotografie a feței tale este capturată dintr-o fotografie sau videoclip. Fața ta ar putea apărea singură sau în mulțime. Imaginea dvs. vă poate arăta direct sau aproape de profil.

Pasul 2. Software-ul de recunoaștere facială citește geometria feței tale. Factorii cheie includ distanța dintre ochi și distanța dintre frunte și bărbie. Software-ul identifică reperele faciale - un sistem identifică 68 dintre ele - care sunt cheia pentru a vă distinge fața. Rezultatul: semnătura ta facială.

Pasul 3. Semnătura ta facială - o formulă matematică - este comparată cu o bază de date cu fețe cunoscute. Și ia în considerare acest lucru: cel puțin 117 milioane de americani au imagini ale fețelor lor într-una sau mai multe baze de date ale poliției. Potrivit unui raport din mai 2018, FBI a avut acces la 412 milioane de imagini faciale pentru căutări.

Pasul 4. Se face o determinare. Amprenta dvs. se poate potrivi cu cea a unei imagini dintr-o bază de date a sistemului de recunoaștere facială.

În general, așa funcționează recunoașterea facială, dar cine o folosește?

Aceasta tehnologie a ajuns sa fie foarte populară și se poate utiliza în orice domeniu, în mare parte fiind utilizata de agențiile de securitate națională (SRI) sau chiar și de politia romana.

Implementare

Pentru implementarea aplicației așa cum am specificat mai sus am folosit mai multe medii de dezvoltare a aplicațiilor.

Backend

Recunoașterea Facială

Pentru partea de backend am decis sa folosit Django deoarece este un framework scris în Python, iar acest limbaj de programare fiind unul interpretat si nu compilat, ne-a ajutat foarte mult sa grăbim procesul de dezvoltare a aplicației.

API-ul are un singur endpoint și acesta fiind ‘face_detection/detect/’. Acest endpoint este unul pe care se se poate folosi doar metoda HTTP POST, deoarece aceasta metoda este cea mai sigură dintre toate.

În cadrul acestui endpoint, se executa mai multe comenzi. Ordinea lor ar fi:

- ☐ Accesam fisierul in care sunt tinute imaginile care reprezinta persoanele care dorim sa le dăm access pe site. Aceste imagini sunt procesate pentru a patra doar un vector cu trăsăturile feței din fiecare imagine.
- ☐ Următorul pas implica sa procesam imaginea primită de la aplicația client. Procesarea se realizeaza in aceeasi maniera ca si procesarea pentru datele deja existente pe server.
- ☐ Ultimul pas este sa comparam poza venită de pe aplicația client cu toate pozele salvate deja la noi în aplicație. Comparant imaginile deja prelucrate. Dacă am găsit-o poza care sa aibă aceeași trăsături cu cea venită de la user atunci vom returna token-ul de autentificare, dacă nu vom returna un mesaj de eroare.

Codul sursa pentru detectarea și recunoașterea facială.

```
@csrf_exempt
def detect(request):
    # initialize the data dictionary to be returned by the request
    data = {"success": False}
    # check to see if this is a post request
    if request.method == "POST":
        # check to see if an image was uploaded
        if request.FILES.get("image", None) is not None:
            # grab the uploaded image
            image = _grab_image(stream=request.FILES["image"])
        # otherwise, assume that a URL was passed in
        else:
            # grab the URL from the request
            url = request.POST.get("url", None)
            # if the URL is None, then return an error
            if url is None:
                data["error"] = "No URL provided."
                return JsonResponse(data)
            # load the image and convert
            image = _grab_image(url=url)

        # Get the training images and make a vector with them encoded
        train_images_path = 'face_detector/TrainImages'
        train_images = []
        class_names = []
        images_file_names_list = os.listdir(train_images_path)
        # myList = ['Billie Eilish.jpeg', 'Barack Obama.jpeg']

        for file_name in images_file_names_list:
            current_image = cv2.imread(f'{train_images_path}/{file_name}')
            train_images.append(current_image)
            class_names.append(os.path.splitext(file_name)[0])
        # classNames = ['Billie Eilish', 'Barack Obama']

        def find_encodings(images):
```

```
        """
            Find encoding will take as a param a list of images and return a list of encoded images
        """
        encode_list = []
        for image_to_encode in images:
            current_img = cv2.cvtColor(image_to_encode, cv2.COLOR_BGR2RGB)
            encode = face_recognition.face_encodings(current_img)[0]
            encode_list.append(encode)
        return encode_list

    encoded_train_images_list = find_encodings(train_images)

    faces_input_photo = face_recognition.face_locations(image)
                encoded_input_photo = face_recognition.face_encodings(image, faces_input_photo)

                    for encode_face, face_location in zip(encoded_input_photo, faces_input_photo):
                matches = face_recognition.compare_faces(encoded_train_images_list, encode_face)
            face_distances = face_recognition.face_distance(encoded_train_images_list, encode_face)
                            # face_distances - array containing how similar ar the encoded_train_images_list with encode_face
            match_index = np.argmin(face_distances)
            if matches[match_index]:
                resulted_name = class_names[match_index].upper()
                data.update({"success": True, "name": resulted_name})
            else:
                data.update({"success": False, "name": 'ERROR'})
        # update the data dictionary with the faces detected
    # return a JSON response
    return JsonResponse(data)


def _grab_image(path=None, stream=None, url=None):
    # if the path is not None, then load the image from disk
    if path is not None:
        image = cv2.imread(path)
    # otherwise, the image does not reside on disk
    else:
        # if the URL is not None, then download the image
        if url is not None:
            resp = urllib.urlopen(url)
            data = resp.read()
        # if the stream is not None, then the image has been uploaded
        elif stream is not None:
            data = stream.read()
        # convert the image to a NumPy array and then read it into
```

```
        # OpenCV format
        image = np.asarray(bytearray(data), dtype="uint8")
        image = cv2.imdecode(image, cv2.IMREAD_COLOR)

    # return the image
    return image
```

Django

Django este un cadru web open source pentru Python. Oferă un nivel înalt abstractizarea modelelor comune de dezvoltare web. Cadrul Django urmează Model de proiectare Model-View-Controller (MVC). Folosește MVC pentru a separa modelul ca date și logică de afaceri a aplicației, vizualizare ca reprezentare a informațiile pentru utilizator, în acest caz, partea clientului a aplicației și controler ca o interfață a aplicației, în acest caz, set de adrese URL la comunică cu front-end

Frontend

Fundamente teoretice

Un prototip timpuriu al React, dezvoltat de un inginer software la Facebook numit Jordan Walke, a fost introdus pentru prima dată în 2011 sub numele de FaxJS. Walke a finalizat prototipul și a creat React în 2012, care a fost în curând integrat în Facebook și Instagram în același an. În 2013 React a devenit open-source și ulterior au devenit disponibile în Ruby on Rails și Python Applications.

Aceasta este urmată de lansarea React Native, o extensie a React pentru mobil dezvoltare pe Android și iOS, în 2015. De atunci reacționează constant împinge multe versiuni de-a lungul anilor, îmbunătățind și introducând noi caracteristici pentru utilizatori. (Hámori 2020.)

Se știe că React este un instrument versatil care poate fi utilizat atât pe desktop, cât și pe desktop platforme mobile cu multe caracteristici distincte. Una dintre ele este capacitatea de a crea pagini interactive, mai degrabă decât simple, care pot actualiza și reda date după fiecare intrare de la utilizator, permite astfel o interfață interioară fără a fi nevoie să reîmprospătați întreaga pagină de fiecare dată când se face o modificare. Motivul pentru aceasta se datorează lui React structura se bazează pe componente care permit proiectarea unor UI complexe datorate logica componentelor este scrisă în JavaScript

Când utilizați mai multe componente într-o aplicație React, este important ca fiecare componentă a clasei are o metodă a ciclului de viață pentru a preveni resursele fiind irosit când componentele sunt distruse. Adăugând o viață metoda

ciclului asigură controlul asupra creării și încetării componente precum și proprietățile lor, cum ar fi când primesc noi proprietăți sau când ar trebui să fie actualizate. Unele cicluri de viață utilizate frecvent metodele includ:

- constructor (props): Această metodă este utilizată la inițializarea primei instanța a componentei, precum și starea inițială a componentei.

- componentDidMount (): Această metodă este utilizată după primul apel de redare și poate fi util pentru accesarea DOM sau pentru efectuarea de cereri HTTP.

- componentWillUnmount (): Această metodă este utilizată înainte ca o componentă să fie este distrus și este utilizat în mod obișnuit pentru a șterge cronometrele sau a anula rețeaua solicitări.

- componentDidUpdate (): Această metodă este utilizată după metoda de redare sau când apare o actualizare, cu excepția primei metode de redare.

Când modificați statul, există trei lucruri de știut pentru a asigura utilizarea corectă a statului. Primul este că statul poate fi modificat numai folosind setState () și, de asemenea, nu pot fi modificate direct. Următorul lucru de știut este că recuzita și starea pot fi actualizate asincron și nu trebuie depinde de aceeași actualizare. Mai multe apeluri setState () pot fi grupate într-un actualizare unică de React pentru a păstra performanța. În cele din urmă, când sunați setState (), React poate îmbina obiectul componentei în starea curentă. Aceasta înseamnă că un stat cu mai multe variabile poate fi actualizat folosind independent mai multe apeluri setState () separate.

Hooks reprezinta cea mai recentă adăugare în React versiunea 16.8, Hooks permite utilizarea Stat fără a fi nevoie de o clasă. În teorie, Hooks sunt funcții care pot să fie utilizat pentru a solicita caracteristici ale stării și ale ciclului de viață din componentele funcției. Există două tipuri de Hooks: Hooks de stat și Hooks de efect. Cârlige de stat sunt folosite pentru a adăuga starea componentelor funcționale. Cârligele de efect sunt obișnuite efectuați efecte secundare în componentele funcției, de exemplu preluarea datelor sau schimbarea manuală a DOM. Hooks sunt compatibile cu versiunile anterioare și pot fi integrat în codul existent alături de clase fără a fi nevoie să descrieți componentele.

Implementare

Pentru aplicația client am decis sa folosim framework-ul React.

Pentru a lua datele pe care sa le prezentam pe site-ul aplicației am folosit pe post de “bază de date” un google spreadsheet. Motivul pentru care am făcut asta este ca, știm că mare parte din rapoartele pe care profesorii trebuie sa le trimita mai departe sunt deja făcute în excel. De aceea am ales sa folosim google sheet, pentru o mai ușoară folosinta a aplicației și pentru un mai bun management al datelor.

Mai jos aveti atasate poze din catrul aplicatiei.

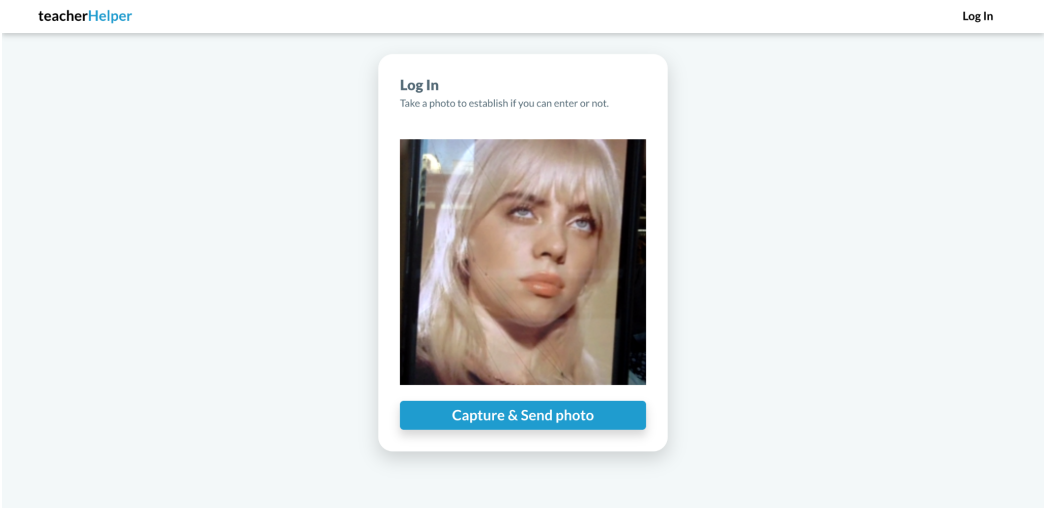

Fig 1: Log in screen

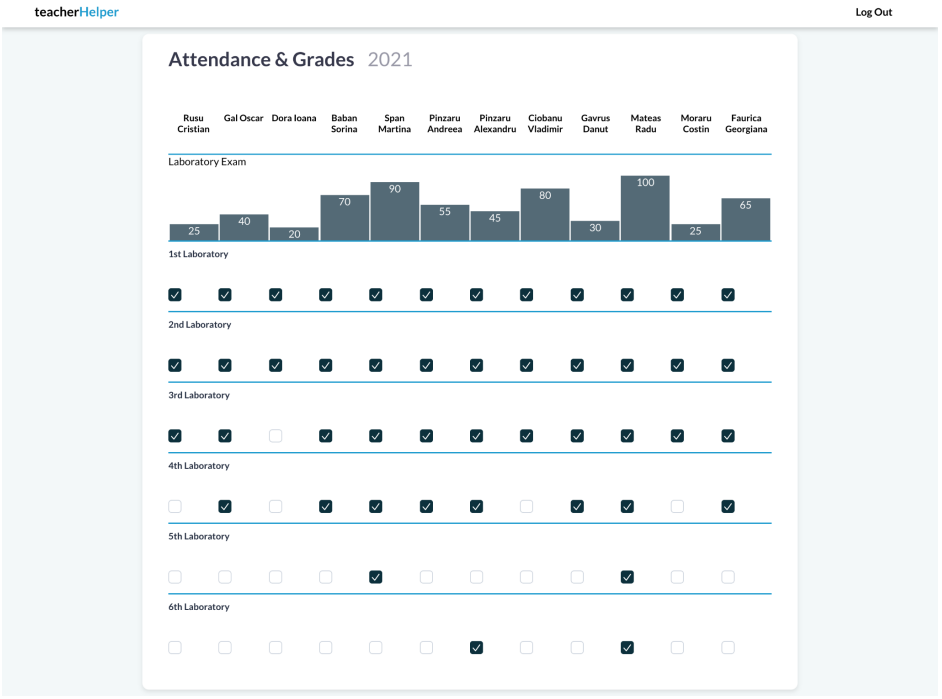

Fig 2 Screen-ul pentru vederea notelor și prezentelor

Următorul cod exemplifica modul în care datele sunt luate de într-un google sheets iar apoi sunt mapate ca sa corespundă cu formatul pe care noi îl dorim.

```
Tabletop.init({
            key: '1R03lJrvAVvssTHwvnDQDyy-n86kkLp4BHtlNNugL2xc',
            simpleSheet: true
        }
    ).then((data) => {
        let result = {year: 2021, students: {}};
        data.forEach((el) => {
```

```
            result.students[el.student] = {
                class1: el.class1 === "TRUE",
                class2: el.class2 === "TRUE",
                class3: el.class3 === "TRUE",
                class4: el.class4 === "TRUE",
                class5: el.class5 === "TRUE",
                class6: el.class6 === "TRUE",
                quiz: el.quiz,
            }
        })
        this.setState({currentReport: {...result}, loading: false})
    })
```

Instalare

Backend

Pentru instalarea aplicației de backend vor fi nevoie mai intai de instalarea limbajului de programare Python3.8. Aveți aici un link pentru instalarea acestui limbaj de programare. https://www.python.org/downloads/

După ce ați instalat limbajul de programare va trebui sa instalati toate dependintele acestui framework, pentru a face acest lucru prima data v-a trebui sa creați un un environnement de dezvoltare utilizând comanda: py -m venv facialauth. După crearea v-a trebui sa și dați drumul acestui environment și acest lucru se va face cu comanda: facialauth\Scripts\activate.bat

Avand environmentul creat și activat acum v-a trebui sa instalam toate dependintele necesare framework-ului sa mearga si pentru a face acest lucru vom folosi comanda: pip install -r requirements.txt.

Toate aceste comenzi au fost necesare pentru pregatirea mediului de dezvoltare al aplicației. Pentru a da drumul acestui server aveți comenzile explicate mai jos:

./manage.py makemigrations - creeaza migrarile pentru baza de date creeata

./ manage.py migrate - aplica aceste migrari

./manage.py createsuperuser - va lasa sa creati un tilizator cu drepturi depline asupra aplicației.

./manage.py runserver - aceasta comanda v-a rula server-ul.

Frontend

Pentru a face aplicația client sa functioneze mai intai vom avea nevoie de cateva framework-uri instalate. Primul lucru care trebuie instalat este node. Node este un manager de biblioteci. Aveți aici link-ul de unde se descarca node și unde explicat și cum se instaleaza. https://nodejs.org/en/download/

După ce a fost instalat node vor trebui instalate și dependințele pentru ca framework-ul React sa functioneze.

Intrați în directorul grades și tastați comanda: npm install. Aceasta comanda v-a instala toate dependintele care apar in package.json

Pentru a rula acest server client se v-a folosi comanda npm start

References

[1] https://face-recognition.readthedocs.io/en/latest/face_recognition.html

[2] https://www.pyimagesearch.com/2015/05/11/creating-a-face-detection-api-with-python-and-opencv-in-just-5-minutes/

[3] https://docs.djangoproject.com/en/3.2/topics/auth/customizing/

[4] Reference. API Blueprint [online]. [Cited 2017-04-22]. Available from: https://apiblueprint.org/

[5] HOLOVATY, Adrian and Jacob. KAPLAN-MOSS. The definitive guide to Django: Web development done right. 2nd ed. Berkeley: Apress, c2009. Expert's voice in Web development., 2009, ISBN 978-1-4302-1936-1.

[6] Levlin, M. 2020. DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte. Åbo Akademi. Master's thesis. Retrieved on 20 October 2020. Available at http://urn.fi/URN:NBN:fife2020051838212