Universitatea "Babeș Bolyai"
Cluj-Napoca

Facultatea de matematică și informatică
Specializare informatică

O soluție mobilă pentru detecția prezenței studenților folosind tehnici de machine learning și computer vision

Machine learning și computer vision

LUCRARE DE LICENȚĂ

Coordonator
Lect. dr. MIRCEA Ioan Gabriel

Autor
ONU Eduard Gabriel

Cluj-Napoca, România
Iulie 2019

Curpins

Lista imaginilor

1 Introducere și motivație

Toate organizațiile folosesc sisteme de prezență pentru a înregistra momentul în care proprii angajați au ajuns sau au plecat de la lucru ori unde și în ce departament aceștia au început să lucreze.

Deoarece un factor important în obținerea unei bune performanțe academice îl are prezența la ore, toate instituțiile academice se concentrează pe obținerea unei valori cât mai ridicate pentru această măsură.

În acest scop, pentru a rezolva această problemă, fiecare instituție a venit cu propria sa soluție: unele pentru a efectua prezența în mod regulat au recurs la strigarea studenților după nume, altele prin semnarea individuală pe hârtii de către studenți, dar aceste metode sunt destul de costistitoare din punct de vedere al timpului alocat atât de elevi cât și de profesori.

Pe de altă parte, în cazul în care avem de-a face cu instituții academice ce au un număr foarte mare de studenți, o astfel de abordare nu oferă un grad ridicat de securitate, acele foi putând fi modificate cu ușurință.

De accea, acest proces ar trebui automatizat, prin introducerea unui sistem de efectuare a prezențelor, un sistem semi-automat în care profesorul să poată observa și să modifice situația prezențelor, ușor de folosit ce are un cost redus și poate oferi rezultate în timp relativ scurt.

1.1 Contribuție personală

Am propus un sistem de prezențe semi-semiautomat ce folosește recunoașterea facială pentru identificarea persoanelor din sala de clasă, sistem alcătuit din două mari componente: **recunoașterea facială** și **detecția facială**.

Acesta, pentru prima parte de detecție facială, folosește rețele neuronale convoluționale iar pentru cea de a doua parte, recunoașterea facială, algoritmul LBPH (Local Binary Pattern Histograms).

Totodată, sistemul propus permite modificarea situației prezențelor, vizualizarea rezultatelor în timp real, este intuititiv, foarte ușor de folosit și personalizat. În plus, se permite înscrierea studenților la diferite cursuri, oferind posibilitatea profesorilor să adauge noi materii.

1.2 Structura lucrării

Pentru o mai bună înțelegere a modului în care soluția a fost implementată, lucrarea este împarțită pe mai multe capitole, în fiecare prezentându-se diferite aspecte ale problemei:

I În **Capitolul 2** se prezintă modurile în care a mai fost abordată această problemă, felul în care aceasta a fost abordată și aspectele pozitive, respectiv cele negative ale fiecărei soluții în parte.

II În **Capitolul 3** se vor introduce noțiunile de bază ce sunt folosite în diferite aspecte ale soluției precum: **rețele neuronale artificiale**, **rețele neuronale artificiale convoluționale**, **algoritmul de recunoaștere facială folosit(LBPH)**, dar și toate celelalte noțiuni necesare înțelegerii complete a termenilor folosiți.

III În **Capitolul 4** se va explica soluția propusă în detaliu de la abstract spre concret. Se vor prezenta diferite diagrame pentru o mai bună înțelegere a structurii soluției, detalia toate tehnologiile folosite și nu în ultimul rând, se va prezenta modul în care aplicația a fost implementată.

IV În **Capitolul 5** se va explica modul în care a fost testat modelul folosit și se vor prezenta rezultatele obținute.

V În **Capitolul 6** se vor menționa problemele întâlnite în dezvoltarea solției și modul în care acestea au fost rezolvate

VI În **Capitolul 7** se vor prezenta concluziile și viitoarele posibile îmbunătățiri

VII În **Capitolul 8** este prezentată bibliografia

2 Abordări înrudite

2.1 Sisteme bazate pe recunoașterea facială

În [2] doi cercetătători, Visar Shehu și Agni Dika au porpus un sistem ce folosește, pentru efectuarea prezențelor atât vederea artificială cât și recunoașterea facială.

Sistemul este implementat folosind o camera digitală, instalată în sala de clasă, ce are ca principal obiectiv realizarea de fotografii ale elevilor, detectarea și extragerea tuturor fețelor din imagini.

După ce toate fețele au fost detectate și extrase, acestea vor fi comparate cu fețele existente din baza de date iar dacă un student este identificat cu succes, o lista cu prezențe este generată și salvată în baza de date.

Însa din aceasta abordare reies anumite probleme precum detecția facială în timp real în medii ce conțin obiecte multiple, probleme cu algoritmii de recunoaștere facială și probleme de natura socială și pedagocică legate de tehicile aplicate.

În [3] a fost propusă, într-un sistem de supraveghere in timp real, o metoda pentru detectarea simultană a capului și a feței ce implică patru caracteristici direcționale (FDF) si analiza diferențială liniara. FDF este una dintre cele mai robuste caracteristici pentru a distinge modele, iar in esență aceasta este alcatuită din patru componente pentru identificarea marginilor (vertical, orizontal, prima diagonală, a doua diagonală).

Având in vedere că această metodă a atins performanța de aproximativ 10 cadre pe secundă pentru detecția facială, ar mai fi nevoie de foarte multe îmbunătățiri.

În [4] a fost implementat un sistem complet automat pentru efectuarea prezențelor ce se dovedește a fi practic, de încredere și elimină pierderea de timp din sisemele tradiționale de prezență.

În acest sistem întalnim 3 actori: studentul, profesorii și administratorul, fiecare având câte un rol bine definit.

- **Studentul** poate să își urmărească situația prezențelor (pentru acest lucru trebuie să se autentifice în prealabil folosind id-ul și parola asociate).

- **Profesorul** poate să țină evidența prezențelor de la cursurile sale(trebuie să fie autentificat in prealabil).

- **Administratorul** introduce toți actorii (studenții, profesorii), toate cursurile și resetează parolele. Pentru detecția facială s-au folosit diferite tehnici precum detecția bazată pe culoare sau PCA (Principal Component Analysis) și pentru extragerea de caracterstici PCA și LDA (Linear Discriminate Analysis). Detecția bazată pe culoare reprezintă identificarea acelor zone din imagine asemanatoare cu pielea umană.

Aceasta abordare însa ridică anumite probleme. Pe de o parte, probleme ce privesc experiența utilizatorului precum faptul că nu exista niciun mecanism prin care profesorul să poată modifica rezultatele prezențelor sau sa poată să își modifice cursurile pe care le predă, iar pe de altă parte numarul ridicat de detecții fals pozitive.

În [**5**] a fost propus un sistem de prezențe automat ce poate funcționa in timp real, având un timp de detecție și recunoaștere foarte bun, bazat pe identitatea facială, alcătuit din doua componente principale: detecția și recunoașterea facială. Rezultatele prezenței automate sunt afișate într-o foaie de lucru excel.

Acest sistem primește ca și date de intrare imagini cu ajutorul unei camere video, ce functionează continuu până când sistemul este oprit. Modulul de detecție păstrează numai informația facială din pozele primite de la camere, iar modulul de recunoaștere facială încearcă să identifice persoanele din aceste imagini și să le înregistreze că fiind prezente.

Pentru detecția facială s-a folosit algoritmul Viola Jones [**6**] împreună cu clasificatorul Haar, în timp ce mecanismul de recunoaștere utilizează PCA și Fast PCA [**7**].

Cu toate că este un sistem ce poate funcționa în timp real, lipsa unei soluții pentru modificarea manuală a prezențelor de către profesori reprezintă un mare minus pentru această abordare.

2.2 Sisteme bazate pe bluetooth

În 2013, Vishal Bhalla et al. [**8**] a propus un sistem care poate să înregistreze prezența cu ajutorul tehnologiei Bluetooth.

În acest sistem, prezențele sunt înregistrate folosind telefonul profesorului. O aplicație ce este instalată pe telefonul cadrului didactic va interoga, prin Bluetooth, telefonul studentului iar prin transferul adresei MAC(Media Access Controll) a telefonului elevului, prezența poate fi memorată și adaugată in aplicație.

Un principal dezavantaj al acestei abordări este acela că nu e nevoie ca studentul să fie prezent în clasă, fiind îndeajuns doar ca un alt student să aibă telefonul acestuia, pentru ca prezența să fie înregistrată.

2.3 Sisteme bazate pe NFC

În [**9**] autorul ne prezinta implementarea unui sistem de prezență care este bazat atât pe Bluetooth cât și pe tehnologii NFC(Near Field Comuication). Utilizează amprenta și adresa bluetooth a telefonului ce are NFC-ul pornit pentru a înregistra identitatea acestuia.

O aplicație scrisă in Java [**10**] recepționează id-urile NFC împreună cu alte informații referitoare la telefonul mobil și utilizator, urmând ca acestea să fie trimise către un analizator pentru interpretarea comportamentului acestuia. O constrângere este aceea că, pentru a folosi acest sistem trebuie ca fiecare student să aibă telefon compatibil cu tehnologia NFC, pentru a i se înregistra prezența, ceea ce implică costuri suplimentare.

2.4 Sisteme bazate pe scanarea irisului

În 2010, Seifedine Kadry și Mohamad Smaili au propus un sistem de prezență care este conceput și implementat utilizând algoritmul lui Daugman. Acest sistem rezolvă problema prezențelor false utilizând tehicii de scanare a irisului și totodată facilitează procesul de efectuare al prezențelor. [**11**]

Însă pricipala problemă pe care un astfel de sistem o ridică este costul foarte mare pentru realizarea sa. Pe lângă această problemă, un student tebuie să aștepte un numar considerabil de secunde pentru a-i fi scanat irisul. [**11**]

2.5 Sisteme bazate pe amprentă

În [**12**], Mohamed et al., a creat un dispozitiv bazat pe amprentă folosit în sistemele de prezență automată. Studenților li se acordă prezența dupâ ce își plasează degetul pe senzorul dispozitivului.

Însă acest sistem nu s-a dovedit a fi așa de folositor deoarece pe de o parte, dispozitivul ce conținea senzorul de amprentă se strica frecvent, iar pe de altă parte, necesită destul de mult timp pentru efectuarea prezenței deoarece studenții trebuiau să astepte la coadă pentru acel dispozitiv.

2.6 Sisteme bazate pe RFID

BIS în [**13**] prezintă un sistem comercial bazat pe RFID (Radio Frequency Identification) de prezentă automată pentru școli si licee. Sistemul poate să trimită mesaje SMS sau alerte email către parinți sau cadre didactice în mod automat. Aceștia se vor înregistra la poartă, atingând dispozitivul RFID cu ecusonul, urmând ca datele să fie trimise către un server din școală. Acest server va procesa datele și va trimite SMS-uri către pariniți.

Principala problemă a acestui sistem constă în faptul că nu este efectuată verificarea situației prezențelor. Totodată, sistemul nu dispune de o metodă pentru modificarea prezențelor acordate în mod automat.

După cum putem observa, cu toate că există o varietate de implementări pentru sistemele de prezență automată, nu există o implementare care să ofere atât o performanță acceptabilă, cât și o bună experiență a utilizatorului la un preț acceptabil. De aceea un sistem de prezențe semiautomat, bazat pe recunoaștere facială, în care profesorul să poată să intervină asupra rezultatelor reprezintă cea mai bună soluție.

3 Fundamente teoretice

3.1 Învățarea automată

Învățarea automată este un domeniu al inteligeței artificiale(AI) ce oferă unor sisteme abilitatea de a învăța și de a se autodepăși fără a necesita programarea în mod explicit. Acest domeniu are ca principal obiectiv dezvoltarea programelor pentru calculatoare ce pot accesa date cu scopul de a învăța.

Procesul de învătare începe cu date precum exemple, experiența directă sau instrucțiuni ce au ca principal obiectiv găsirea unor modele în date și de efectuarea unor predicții bazate pe experiențele din trecut.

Acești algoritmi vor produce o funcție ce are ca principal obiectiv identificarea unei legături între datele de intrare și rezultatele obținute de la sistem. Această funcție se numește **model**. [**14**]

Există trei tipuri de modele:

- **Modele geometrice**: acestea pot avea 1, 2 sau mai multe dimensiuni și sunt învațate din date. Deseori sunt folosite în probleme în care dorim să clasificăm datele în categorii.
- **Modele probabilistice**: acestea încearca să determine distribuția probabilistică a datelor, rezultatul lor fiind de obicei o valoare intre 0 și 1.
- **Modele logice**: acestea încearca din punct de vedere logic să ofere un rezultat. Domeniul de valori este discret, find deseori etichete.

1. **Învățarea supervizată**

   Se cunosc atât datele de antrenament cât și etichetele acestora.

   Se va învăța din exemple și se va produce o funcție capabilă să facă predicții despre datele de ieșire, însă deseori pentru rezultate cât mai bune este nevoie de un număr foarte mare de date, ceea ce cateodată nu este posibil. [**14**]

2. **Învățarea nesupervizată**

Este utilizată atunci când datele nu sunt nici calsificate nici etichetate. Învătarea nesupervizată studiază modul în care sistemele pot determina o modalitate de descriere a structurile ascunse din datele neetichetate.[**14**]

3. **Învățarea semi-supervizată**

   Aceasta se încadrează între învățarea supervizată și cea nesupervizată deoarece se folosesc atât date etichetate cât și date neetichetate pentru antrenare. De obicei se utilizează un număr foarte mare de date neetichetate și un număr relativ mic de date etichetate.

   Algoritmii care utilizează învățarea semi-supervizată sunt folosiți când este nevoie să se rezolve o problemă ce necesită date ce nu pot fi etichetate foarte ușor, fiind nevoie de specialiști în domeniu pentru a realiza acest lucru.[**14**]

4. **Învățarea prin întarire**

   Aceasta este o metoda de învățare în care se interacționează cu mediul, permițând mașinilor și programelor să determine în mod automat comportamentul ideal într-un context specific cu scopul de a maximiza câstigul. Este introdus conceptul de recompensă și penalizare, acestea fiind utilizate în procesul de antrenare al agentului.[**14**]

De-a lungul timpului, pentru fiecare tip de învățare au fost dezvoltați și implementați diferiți algoritmi de inteligență artificială, aceștia împarțindu-se in două mari categorii: **algoritmi de învățare supervizați** și **algoritmi de învățare nesupervizați**.

3.2 Rețele neuronale artificiale

Rețelele neuronale artificiale(figura 1) reprezintă o serie de algoritmi, inspirați din modul de funcționare al creierului uman, ce au ca principal obiectiv identificarea de tipare în date. „Tiparele" pe care acestea le recunosc sunt de natură numerică, reprezentate sub forma de vectori iar de accea indiferent că lucrăm cu sunete, text sau imagini singurul mod în care rețeaua poate să interacționeze cu aceste date este translatarea acestora în vectori. [**15**]

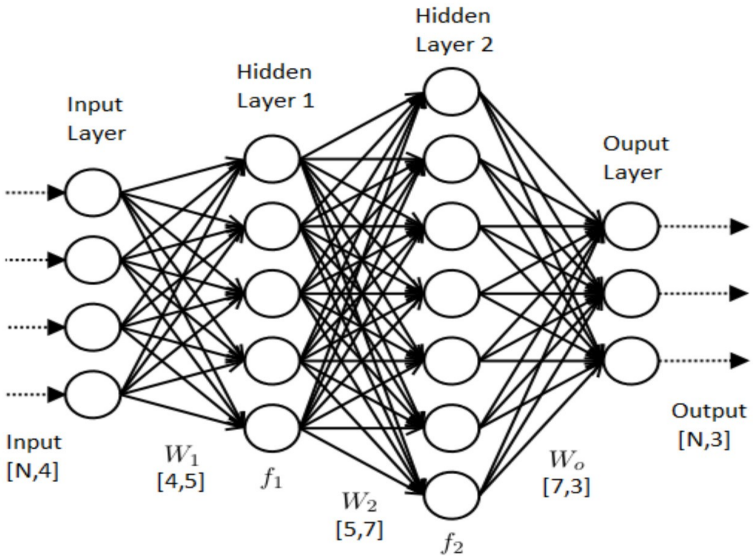


Figure 1: Neuronal network [**15**]

Rețelele neuronale pot fi folosite cu success in probleme de **clasificare**, **grupare de date** și in **analize predictive**.

O rețea neuronală este alcatuită din mai multe straturi de neuroni, adică acele componente ce se ocupă de procesarea datelor.

3.2.1 Neuronul artificial

Neuronul artificial combină datele de intrare cu un set de coeficienti sau ponderi, care amplifică sau micșorează cantitatea fiecărui parametru al acestor date.

Acest procedeu de ajustare a ponderilor din rețea, este echivalentul artificial al învațării, reprezentând procesul principal ce stă la baza funcționării unei rețele neuronale. Aceste ponderi de intrare sunt adunate, iar suma este trimisă mai departe spre o **funcție de activare** (figura 2), cu scopul de a vedea dacă și in ce măsură acel semnal ar trebui trimis mai departe, iar in acest caz se spune că neuronul a fost activat.

O rețea neuronală este alcatuită dintr-un **strat de intrare**, unul sau mai multe **straturi ascunse** și un **strat de ieșire**, fiecare dintre aceste straturi folosind datele obținute din stratul anterior, exceptând stratul de intrare.

Având în vedere că operațiile de pe același strat sunt independente, calculele pot fi paralelizate, folosind un GPU, pentru a accelera procesul de învatare al rețelei.

Rețelele neuronale profunde se deosebesc de rețelele neuronale clasice, cu un singur strat ascuns, prin profunzimea lor, mai precis prin numărul de straturi prin care datele

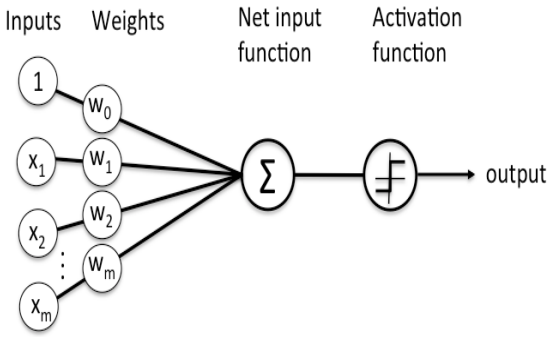


Figure 2: Procesul de activare al unui neuron [**15**]

trebuie să treacă intr-un proces de identificare al tiparelor din date.

Cu cât avansăm mai mult in interiorul rețelei neuronale, cu atât caracteristicile identificate sunt mai complexe deoarece acestea recombină și agregă caracteristicile obținute de straturile anterioare.

3.2.2 Funcția de eroare

Această funcție are ca principal obiectiv măsurarea performanței algoritmului.

Cu cât eroarea este mai mică cu atât rețeaua modelează mai bine setul de date iar din această cauză scopul antrenării unei rețele neuronale este minimizarea funcției de eroare (întalnită sub numele de funcție de loss). Printre cele mai comune funcții de eroare amintim [**16**]:

- Mean Squared Error sau Quadratic Loss

$$MSE = \frac{\sum_{i=0}^{n}(y_i - Y_i)^2}{n}\textbf{[16]} \tag{1}$$

  Este măsurată ca media diferenței pătratice dintre predicția rețelei si rezultatul așteptat.

  Aceasta se concentrează pe magnitudinea medie a erorii, neținând cont de direcție. Totuși din cauza ridicării la putere, predicțiile ce diferă mult sunt penalizate foarte tare în comparație cu cele ce sunt mai apropiate de rezultatul așteptat.

- Mean Absolute Error

$$MAE = \frac{\sum_{i=0}^{n}|y_i - Y_i|}{n}\textbf{[16]} \tag{2}$$

Este măsurată ca media diferenței dintre predicția rețelei și rezultatul așteptat.

Această masura este semanatoare cu (1) doar că necesită instrumente mai avansate pentru calcularea gradienților. Spre deosebire de (1), este mai robustă la valorile extreme deoarece nu utilizează ridicarea la pătrat.

- Cross Entropy Loss

$$CrossEntropyLoss = -(y_i \log(Y_i) + (1 - y_i) \log(1 - Y_i))[\mathbf{16}] \tag{3}$$

Este deseori folosită în probleme de clasificare.

Pentru minimizarea acestei funcții de eroare, se folosește așa numita tehnica a **gradientului descrescător**.

3.2.3 Funcția de activare

Fiecare neuron poate să modifice sau nu informația primită, cu ajutorul funcției de activare. Există mai multe tipuri de funcții de activare [**17**] printre: **funcția liniara** (fig.3), **funcția sigmoid**(fig.4), **ReLu** (fig.5), **tanh** (fig.6).

Din nefericire nu există un algoritm cu ajutorul căruia să alegem funcția de activare ce se portivește cel mai bine tipului nostru de problemă. [**17**]

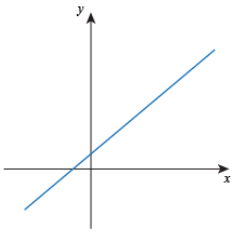


Figure 3: Funcția liniara [**17**]

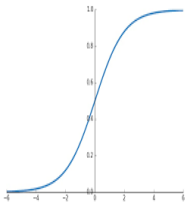


Figure 4: Funcția sigmoid [**17**]

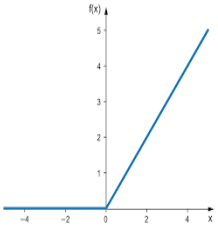


Figure 5: Funcția ReLu [**17**]

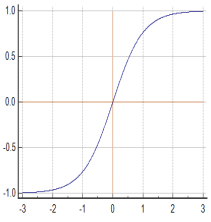


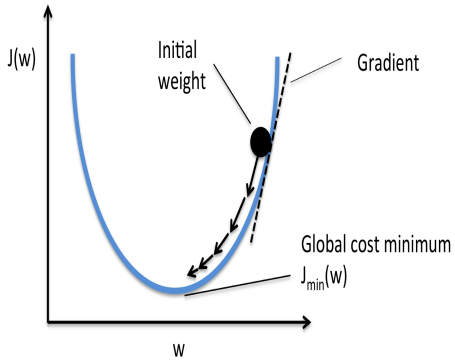
Figure 6: Funcția Tanh [**17**]

3.2.4 Tehnica gradientului descrescător

Acest proces reprezintă tehnica prin care ponderile rețelei sunt ajustate în concordanță cu eroarea. Gradientul nu este altceva decât un alt termen folosit pentru cuvantul „pantă" și reprezintă modul în care două variabile se influentează reciproc.

În cazul unei rețele neuronale ne interesează „panta" ce descrie relația dintre aceasta și o anume pondere, iar ceea ce ne dorim este să aflăm cât de mult înfluențează acea pondere valoarea erorii.

Pe măsură ce rețeaua neuronală învață, se vor ajusta treptat foarte multe ponderi, iar relația dintre **eroarea rețelei** și acele ponderi este o derivată (a erorii in raport cu ponderea) ce măsoara cum o schimbare mică a ponderii cauzează o schimbare mică a erorii.



$$\frac{\partial Error}{\partial pondere} = \frac{\partial Error}{\partial activation} * \frac{\partial activation}{\partial pondere}$$

$$pondere_j = pondere_j - \alpha * \frac{\partial Error}{\partial pondere_j}(pondere)$$

Figure 7: Formula de ajustare a ponderilor

Unde α din (fig. 7) reprezintă rata de învătare, adică acel număr ce indică cât de mult să ne deplasăm spre minimul funcției, iar Error este funcția de eroare.

Totuși, dacă rata de învățare are o valoare foarte mare putem să sărim peste valoare minimă iar rețeaua să nu fie capabilă să învețe. Acest procedeu de actualizare a parametrilor în funcție de output poartă numele de back-propagation.

Pentru rețele cu un număr foarte mare de straturi ce au anumite funcții de activare este totuși posibil ca învătarea să decurgă foate lent din cauza faptului că gradienții funcției de eroare se pot apropia de valoarea 0. Această problema este intalnită sub numele de **problema gradientului ce dispare**(vanishing gradient).

Un alt factor ce ar putea duce la probleme de învățare este **problema gradientului ce explodează** (exploding gradient).

În estență este fix problema inversă celei menționate mai sus unde de această dată,

din cauză că valorile ponderilor pot să ajungă foarte mari, acestea pot să nu mai încapă într-o reprezentare în virgulă flotantă(fenomenul de depașire). Pentru evitarea acestor probleme s-au propus diferiți optimizatori pentru algoritmul de gradient descrescător.

3.2.5 Optimizatori pentru algoritmul de gradient descrescător

Există foarte mulți optimizatori pentru algoritmul de gradient de descrescător iar în această lucrare se vor prezenta doi dintre aceștia: **RMSprop** și **Adam**

- **RMSProp**

  Root Mean Square Prop sau RMSProp este un algoritm care utilizează același concept ca și gradientul descrescător cu impuls [**18**] și anume media ponderata a gradienților .

  Modul de calculare al gradiențiilor diferă intre gradient descent și RMSProp [**19**]:

$$v_{dw} = v_{dw} * \beta + (1 - \beta) * dw$$

$$v_{db} = v_{db} * \beta + (1 - \beta) * db$$

$$W = W - \alpha * v_{dw}$$

$$b = b - \alpha * v_{db}$$

Modul de calcularea al gradienților in gradient descent [**19**]

$$v_{dw} = v_{dw} * \beta + (1 - \beta) * dw^2$$

$$v_{db} = v_{db} * \beta + (1 - \beta) * db^2$$

$$W = W - \alpha * \frac{v_{dw}}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha * \frac{v_{db}}{\sqrt{v_{db}} + \epsilon}$$

Modul de calculare al gradienților in RMSProp [**19**]

  Așa cum putem observa, acest optimizator restricționează oscilațiile gradientului în direcția verticală asfel incât algoritmul să conveargă mai repede.

- **Adam**

  Acest algoritm de optimizare combină avantajele algoritmului RMSProp cu AdaGrad într-o singură implementare. În loc să adapteze rata de învățare a fiecărui parametru bazat pe media primului impuls precum RMSProp, acesta utilizează media impulsurilor secundare a gradienților.

Este printre cei mai folosiți algoritmi de optimizare datorită faptului că acesta obține rezultate foarte bune, într-un timp relativ scurt .

3.3 Rețele neuronale artificiale convoluționale

În învățarea profundă, o rețea neuronală convoluțională (CNN sau ConvNet) este o clasă a rețelelor neuronale profunde, aplicată deseori pe probleme de analiză a imaginilor. Deși conceptul este relativ vechi, acestea au fost inventate în anii 1980, CNN-urile au început să fie folosite de abia în ultimii ani, acest lucru fiind datorat creșterii puterii de procesare al plăcilor video(GPU) [**20**].

În esență, o astfel de rețea este alcătuită din mai multe straturi de neuroni artificiali.

Întâlnim straturi de intrare, de ieșire, fully-connected, straturi de pooling, de dropout și de convoluție. Acestea din urmă reprezintă conceptul ce diferențează o rețea neuronală artificială convoluțională de rețelele neuronale artificiale clasice (ANN). [**20**].

- **Statul de convoluție (convoluțional layer)** [**21**]
  - Reprezintă stratul prin care o rețea neuronală convoluțională se diferențează de o rețea neuronală clasică. Scopul acestui strat este identificarea unor caracteristici ale obiectelor în imagine.

    Prima oară când o imagine este prezentată unei rețele neuronale aceasta nu știe „după ce să se uite", dar cu timpul aceasta va învăța anumite filtre, pe care le va aplica pe imagine, pentru a găsi acele caracteristici ce definesc obiectul căutat.

- **Stratul de intrare (input layer)** (identic ca la ANN) [**21**]
  - Stratul de intrare poate fi văzut drept condiția pentru care noi antrenăm rețeaua neuronală.

    Fiecare neuron poate fi văzut ca o variabilă independentă ce are o foarte mare influență asupra rezultatului obținut de rețeaua neuronală.

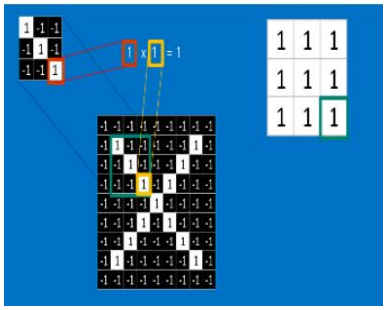


Figure 8: Exemplu de filtru [**21**]

Un filtru (fig. 8) este o matrice cu valori de 1 și -1. Convoluția reprezintă înmulțirea element cu element a numerelor din filtru cu pixelul coresponedent din imagine, adunarea acestor numere și împărțirea lor la numărul total de elemente din filtru. Se obține astfel o nouă matrice cu valori între -1 și 1.

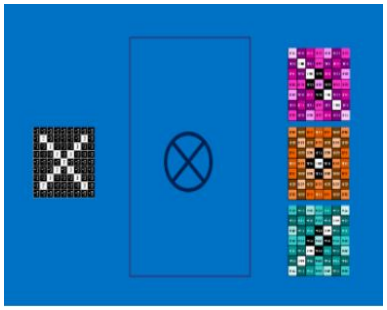


Figure 9: Aplicarea diferitelor filtre [**21**]

Acest procedeu se repetă (fig. 9) până când este acoperită toată matricea, după care același lucru se recalculează și cu alte filtre rezultând mai multe matrici, o matrice per filtru aplicat. Fiecare element al acelei matrici reprezintă procentajul de potrivire al caracteristicii căutate (cu ajutorul filtrului) în acea bucățică din imagine. Datorită acestui proces, fiecare strat de convoluție mărește numărul de imagini ce trebuie să fie procesat de următorul strat din rețea, iar din această cauză procesul de învățare a acestor rețele este atât de lent. Valorile negative se elimină cu ajutorul funcției ReLu (fig. 5)

Acest strat comunică cu mediul extern și are ca principal obiectiv recepționarea informației și transmiterea acesteia către straturile următoare.

- **Stratul de iesire (output layer)** (identic ca la ANN) [**21**]
  - Stratul de ieșire colectează informația primită de la straturile anterioare și o transmite din rețeaua neuronală mai departe spre mediul exterior, într-un mod ce depinde de tipul de problemă rezolvată. Acest strat poate să transmită fie un număr, o etichetă, o listă de numere, practic, orice, depinzând în totalitate de problema pe care o rezolvă rețeaua neuronală.

Modelul identificat de acest strat poate fi întors înapoi către stratul de intrare. Numărul neuronilor din acest strat trebuie să fie corespunzător cu tipul de problemă pe care rețeaua neuronală îl rezolvă.

- **Stratul ascuns (hidden layer)** (identic ca la ANN) [**21**]
  - Stratul ascuns este un strat intermediar ce este alcătuit dintr-o colecție de neuroni, fiecare neuron având câte o funcție de activare. Scopul său este să procesese informația primită de la stratul anterior adică să găsească și să extragă caracteristici din aceste date.

- **Stratul de unificare (pooling layer)** [**21**]

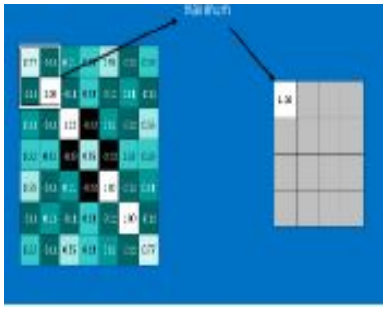


Figure 10: Pooling [**21**]

Acest strat are ca principal obiectiv micșorarea imaginii, păstrând cea mai relevantă informație din acestea. Acest procedeu este realizat prin "glisarea" unei matrice de dimensiune 2x2 sau 3x3 de-a lungul imaginii și calcularea valorii maxime din acea porțiune. Doar valoarea maximă va fi salvată în noua imagine, ce va avea dimenisune de 2, respectiv 3 ori mai mică.

- **Stratul de regularizare (dropout layer)** [**22**]

Acest strat are ca obiectiv principal evitarea învătarii mecanice într-o rețea neuronală.

Acest lucru se realizează prin „ignorarea" unor neuroni în procesul de propagare inversă a erorii.

Folosirea startului de regularizare dublează numărul de iterații necesare pentru a converge, dar acest proces forțează reteaua să învețe caracteristici mai robuste.

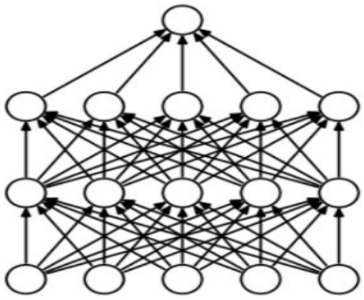


Figure 11: Înainte de a efectua regularizarea [**22**]

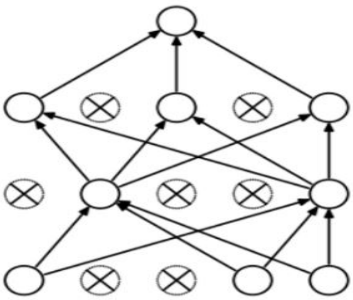


Figure 12: După ce s-a efectuat regularizarea [**22**]

3.3.1 Metode de măsurare a performanței unei rețele neuronale

Pentru măsurarea performanței unei rețele neuronale s-au introdus, de a lungul timpului mai multe metrici, printre care se număra:

- **Măsura recall [23]**

  Folosită pentru măsurarea performanței unei rețele neuronale. Încearcă să identifice ce proporție din actualele pozitive au fost identificate în mod corect.

  Se poate calcula dupa următoarea formulă:

  $$Recall = \frac{TP}{TP + FN} \tag{4}$$

  TP reprezintă valorile adevarat-pozitive, FN reprezintă valorile fals-negative

- **Măsura precision [23]**

  Folosită pentru a măsura procentajul de exactitate al predicției făcute.

  Se poate calcula dupa următoarea formulă:

  $$Precision = \frac{TP}{TP + FP} \tag{5}$$

  FP reprezintă valorile fals-pozitive

- **Măsura mean average precision [23]**

  Se poate calcula ca aria de sub graficul fucției p(r), valoarea acesteia fiind:

  $$AP = \int_0^1 p(r) \tag{6}$$

3.4 Algoritmul de recunoaștere facială LPBH

Recunoașterea facială este procesul prin care o imagie este asociată cu o etichetă unică pe baza caracteristilor indentificate în aceasta. Oamenii, fară efort, efectuează zilnic un astfel de proces iar pe cât se pare că este de ușor pentru noi, pe atât de greu este pentru un sistem soft.

Algoritmul LBPH [**24**] (Locally Binary Pattern Histograms) este încă un operator de textură foarte eficient ce etichetează pixelii din imagine prin aplicarea unui prag în vecinătatea fiecărui pixel și întoarce rezultatul sub forma unui numar binar. [**25**]

S-a constatat că dacă algoritmul clasic (LBP) este combinat cu descriptorul de histograme pe gradienți (HOG [**26**]) se va obține o performanță considerabil mai bună pe anumite seturi de date. Fiind un algoritm, acesta poate fi descris sub forma unor succesiuni de pasi. [**25**]

1. **Setarea parametrilor**

   (a) **Raza**: aceasta este folosită pentru a construi modelul circular local binar și reprezintă raza în jurul pixelului central. De obicei este setat cu valoarea 1. [**25**]

   (b) **Vecinii**: numărul de puncte necesare pentru a construi modelul circular binar. Cu cât acest număr este mai mare cu atât constul de procesare crește. De obicei acesta are valoarea 8. [**25**]

   (c) **Dimensiunea grilei pe X (grid X)**: numărul de celule in direcția orizontală. Cu cât avem un număr mai mare de celule cu atât grila este mai fină dar crește și dimensiunea vectorului de caracterisitici. [**25**]

   (d) **Dimensiunea grilei pe Y (grid Y)**: numărul de celule in direcția verticală. Cu cât avem un număr mai mare de celule cu atât grila este mai fină dar crește și dimensiunea vectorului de caracterisitici. [**25**]

2. **Antrenarea algoritmului**: pentru a obține rezultate, avem nevoie să antrenam algoritmul cu imagini ale persoanelelor pe care dorim să le detectăm. De asemenea, trebuie setat un identificator pentru fiecare imagine, iar imaginile ce aparțin aceleași persoane trebuie să aibă același identificator.

3. **Aplicarea operatiei LBP**: acesta este primul pas ce necesită calcule din algoritmul LBPH(fig. 13) și constă în crearea unei imagini intermediare ce descrie imaginea inițială intr-un mod mai sugestiv pentru algoritm, punând în evidență caracteristicile faciale.

   Pentru a face acest lucru, algoritmul folosește tehnica ferestrei glisante în care se folosesc cei doi parametrii menționați anterior(**rază** și **vecini**). [**25**]

   Se împarte imaginea în subimagini de dimensiunea ferestrei și se alege ca și prag valoarea din mijloc. Aceasta va fi folosită pentru a determina valorile vecinilor (1 dacă valoarea este mai mare decât cea a pragului si 0 alfel).

   După concatenarea acestor valori binare se va obține un număr întreg ce reprezintă valoarea pixelului central din noua imagine. [**25**]

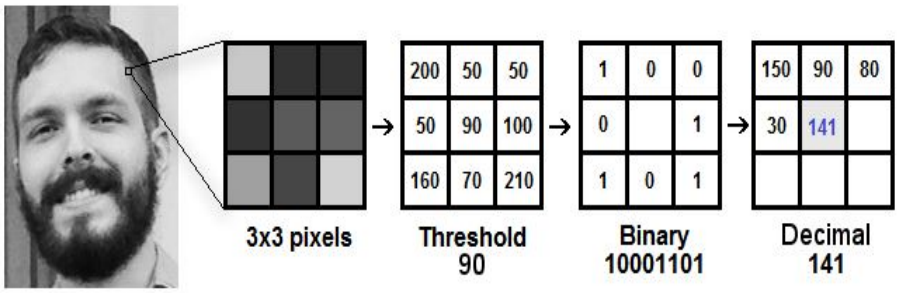


Figure 13: Descrierea procesului [**25**]

4. **Extragerea histogramelor**

   Imaginea generată la pasul anterior va fi împărțită pe baza parametrilor (**grid X** și **grid Y**) în regiuni (fig. 14) iar apoi pe fiecare regiune în parte vom calcula histogramele (numărul de apariții a fiecărei culoare a pixelilor).

   Ulterior, se vor concatena toate aceaste histograme obținând astfel caracterisiticile imaginii originale.[**25**]

5. **Efectuarea recunoașterii faciale**

   Vom repeta din nou pașii anteriori pentru a creea o histogramă pentru această imagine. După aceea vom parcurge setul de histograme folosite pentru antrenare și o vom alege pe aceea care este la distanța cea mai mică de ea.

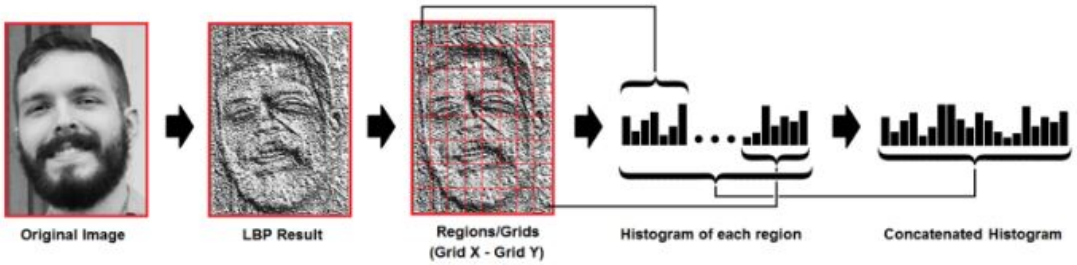


Figure 14: Împărțirea imaginii în regiuni [**25**]

Se pot folosi diferite distanțe printre care și distanța euclidiană.[**25**].

Algoritmul va întoarce identificatorul imaginii și un număr ce reprezintă nivelul de încredere al predicției(care nu este nimic altceva decât distanța dintre cea mai apropiată histogramă și histograma pozei curente).

Cu cât acel număr este mai mic, cu atât predicția facută este mai exactă.

4 Soluția propusă

4.1 Prezentarea ideii

Am propus un sistem (fig.15) ce poate să vină în ajutorul tutoror problemelor menționate anterior, un sistem semi-automat de prezențe bazat pe recunoașterea facială, simplu și ușor de folosit, în care studenții își pot alege la ce materii vor să fie înregistrați de sistem pentru a li se acorda prezențele, cu ajutorul procesului de înrolare la o materie, primesc notificări în timp real, odată ce un proces de prezență automată a fost finalizat și totodată pot vedea un scurt istoric al prezențelor la fiecare materie la care s-au înscris, atât la seminar, laborator cât și la curs. Profesorii pot alege ce materii doresc să predea și au control total asupra prezențelor efectuate de sistem putând să elimine atât prezențele false cât și să adauge persoane neidentificate.

Inițial un utlizator își va face cont pentru a se putea autentifica în aplicație, prin completarea unui formular (nume de utilizator, parola și confirmarea parolei), iar după ce administratorul îi confirmă contul (fie ca profesor, fie ca student), acesta are acces la aplicația propriu-zisă.

În cazul în care contul creat este un cont de tip student, după autentificare, studentul trebuie să aleagă acele materii la care dorește să se înscrie, dar înainte de aceasta, utilizatorul trebuie să încarce două videoclipuri (de maxim 10 secunde fiecare) ce conțin imagini cu fața sa din diferite unghiuri.

Aceste videoclipuri vor fi utilizate pentru extragerea informațiilor legate de fața utilizatorului, în diferite poziții, cât mai diversificate, ce vor fi memorate într-o bază de date, fiind folosite în antrenarea modulului de recunoaștere facială.

După ce procesul de încărcare de videoclipuri, a fost finalizat cu succes, acesta se poate întoarce către pagina principală, unde i se vor arăta rezultatele prezențelor la toate materiile la care s-a înrolat, bineînțeles dacă există rezultate pentru acea materie.

În cazul în care contul creat este de tip profesor, acesta poate să adauge noi materii iar apoi să selecteze cu ce grupa și la ce materie dorește să efectueze prezența, urmând mai apoi să încarce un videclip(de lungime maxim 10 secunde) cu clasa.

Din acest videoclip vor fi extrase persoanele existente, iar după accea pe baza modului de recunoaștere facială se vor identifica studenții in parte, urmând mai apoi ca modulul de prezență automată să folosească aceste informații pentru a înregistra o prezență pentru

fiecare student. Ulterior, după ce procesul se termină, un raspuns va fi trimis atât către fiecare student identificat cât și către profesorul ce a încărcat videoclipul.

După ce a primit rezultatul prezenței, profesorul poate să modifice persoanele absente sau prezente, fiecare modificare declanșând câte o notificare către acel student.

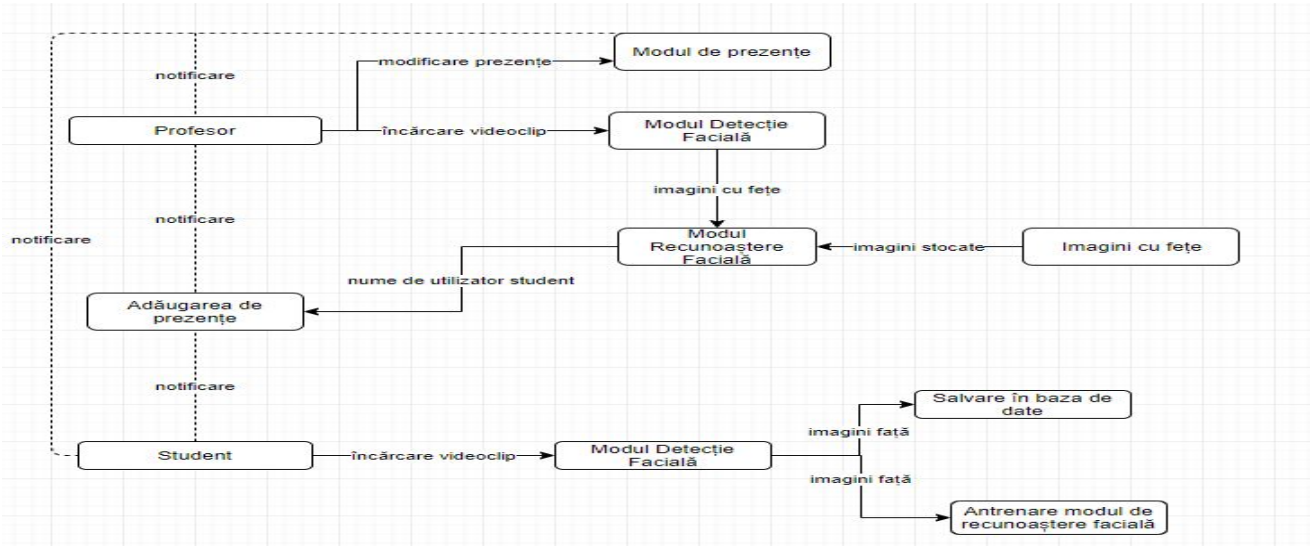


Figure 15: Pașii principali ai sistemului de prezență automată

4.2 Implementarea ideii sub forma unui API

Sistemul are la bază componente critice, fară de care acesta nu ar putea să funcționeze. Aceste componente pot fi folosite indiferent de context, căci reprezintă niște apeluri de metode (API [**27**]) ce se află pe o altă mașină (un server).

În următoarele secțiuni sunt descrise cele mai importante API-uri ale sistemului, modul în care acestea primesc datele de intrare și ceea ce întorc ca răspuns.

4.2.1 API pentru înregistrare (POST)

Calea de acces(endpoint): /auth/register

Acesta este folosit atunci cand un utilizator dorește să își creeaze un cont nou. Primește ca date de intrare un obiect de tip JSON[**28**] cu doua câmpuri: **usern** și **passwd**.

Câmpul **usern** reprezintă numele de utilizator pe care actorul l-a introdus iar **passwd** reprezintă parola.

Acest API întoarce un JSON cu două campuri ca și raspuns: **status** și **message**. Valoarea pe care status o poate avea este fie 200 fie 500, în funcție dacă datele introduse sunt sau nu corecte și dacă nu există in baza de date un utilizator ce are același nume ca și cel introdus, mesajul fiind setat corespunzător.

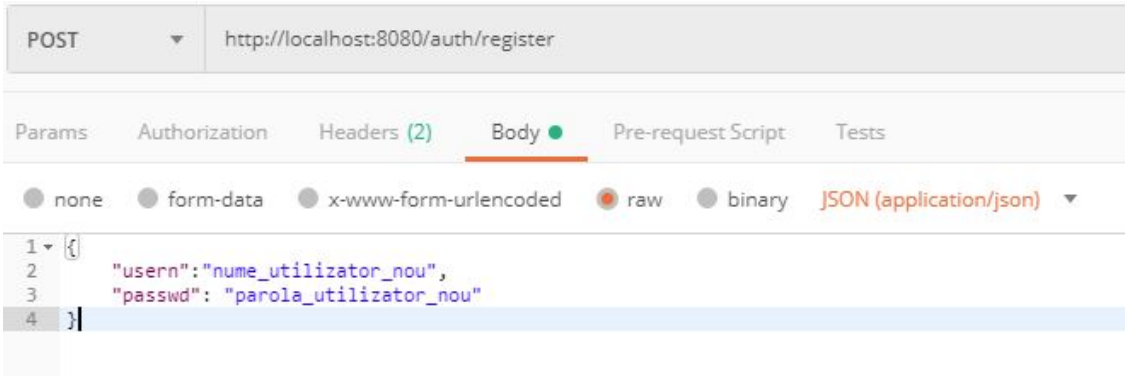


Figure 16: Apelul api-ului de înregistrare

4.2.2 API pentru authentificare (POST)

Calea de acces(endpoint): /auth/login

Acesta este folosit atunci cand un utilizator dorește să se autentifice in aplicație. Primește ca date de intrare un obiect de tip JSON[**28**] cu două câmpuri: **usern** și **passwd**.

Câmpul **usern** reprezintă numele de utilizator pe care actorul l-a introdus iar **passwd** reprezintă parola.

Acest API întoarce un JSON cu două câmpuri ca și raspuns: **status** și **message**. Valoarea pe care status o poate avea este fie 200 fie 404, în funcție dacă datele introduse sunt sau nu corecte, iar mesajul este setat corespunzător.

4.2.3 API pentru încărcarea de videoclip cu poze ce conțin fața (POST)

Calea de acces(endpoint): /stream/update-left-right

Acest api este folosit în momentul în care un student dorește să încarce imagini cu fața sa, pentru a putea fi recunoscut de către sistem.

Primește datele de intrare sub forma unui form, ce conține 3 câmpuri obligatorii: **fileLeftRight**, **user**, **fileUpDown**, acestea reprezentând pe rând, imaginile cu fața in

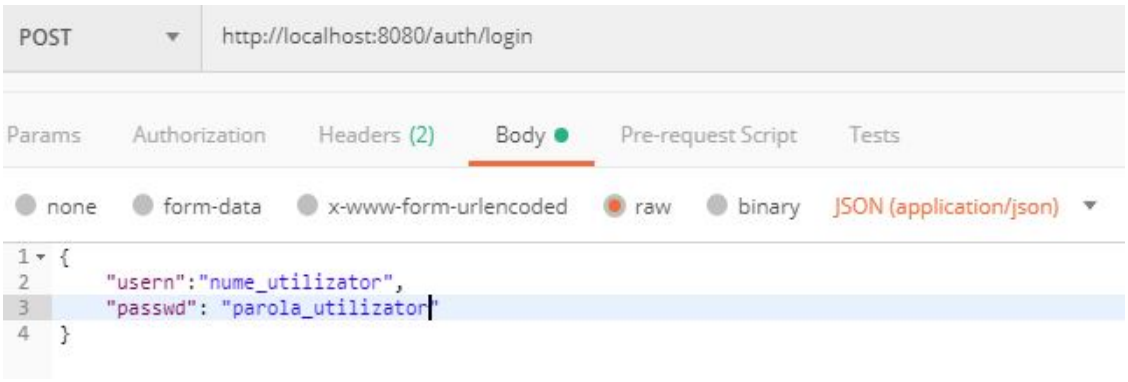


Figure 17: Apelul api-ului de authentificare

momentul în care utilizatorul mișcă capul stanga-dreapta, numele utilizatorului și imaginile cu fața in momentul în care utilizatorul mișcă capul sus-jos.

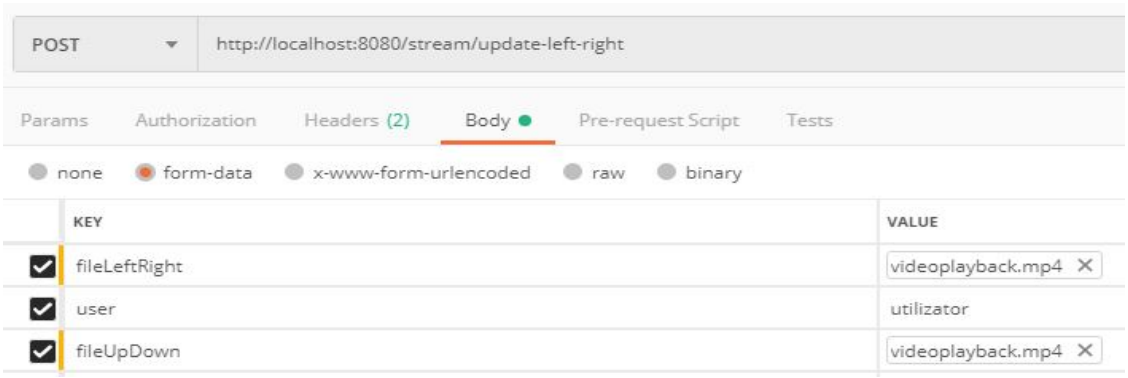


Figure 18: Apelul api-ului de încărcare de poză

4.2.4 API pentru înscrierea la o materie (POST)

Calea de acces(endpoint): /enrollment/add Acest api este folosit în momentul în care un student dorește să de înscrie la o materie predată de un anumit profesor. Primește ca date de intrare un obiect de tip JSON[**28**] cu cinci câmpuri:

- **student**: Numele de utilizator al studentului ce dorește să se înscrie
- **coursType**: Tipul cursului la care vrea să se înscrie, poate fi (SEMINAR, COURSE, LABORATORY)
- **teacher**: Numele de utilizator al profesorului ce a propus materia

- **courseName**: Numele cursului la care dorește să se înscrie
- **group**: Grupa din care face parte studentul

Rezultatul este fie 200, dacă totul este ok, fie 500 dacă unul din câmpuri nu este introdus corect.

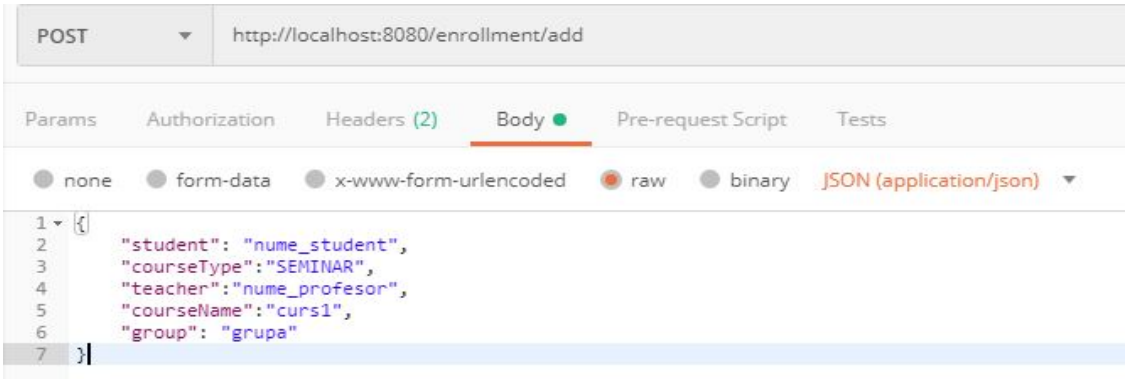


Figure 19: Apelul api-ului de înscriere la o materie

4.2.5 API pentru a aduce istoricul prezențelor a unui student la o materie (POST)

Calea de acces(endpoint): /attendance/for-at

Acest api este folosit pentru a aduce istoricul prezentelor la o materie.

Primește ca date de intrare un obiect de tip JSON[**28**] cu patru câmpuri:

- **student**: Numele de utilizator al studentului ce dorește să se înscrie
- **coursType**: Tipul cursului, poate fi (SEMINAR, COURSE, LABORATORY)
- **teacher**: Numele de utilizator al profesorului ce a propus materia
- **courseName**: Numele cursului

Rezultatul este o listă de prezente (fig.21)

Figure 20: Apelul pentru istoric

Figure 21: Rezultatul apelului

4.2.6 API pentru a adauga o materie (POST)

Calea de acces(endpoint): /courses/add

Acest api este de profesori, pentru a adauga o materie nouă.

Primește ca date de intrare un obiect de tip JSON[**28**] cu patru câmpuri:

- **abr**: Abrevierea cursului
- **coursType**: Tipul cursului, poate fi (SEMINAR, COURSE, LABORATORY)

- **teacher**: Numele de utilizator al profesorului ce a propus materia
- **courseName**: Numele cursului

Rezultatul este fie 200 dacă adaugarea a fost realizată cu succes, respectiv 500 în caz contrar.

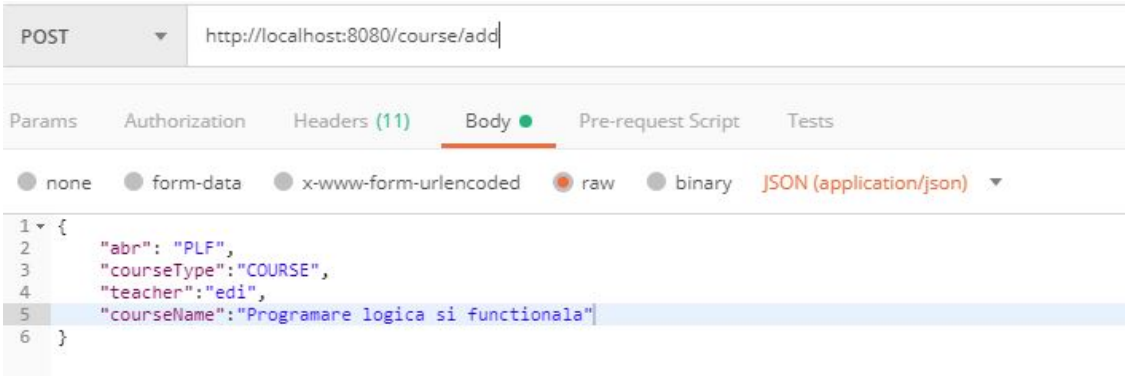


Figure 22: Apelul pentru adăugarea unei materii

4.2.7 API pentru a încărca poza clasei și a efectua prezența automată (POST)

Calea de acces(endpoint): /stream/attendance-video

Acest api este destinat profesorilor, în momentul în care aceștia doresc să facă prezența.

Primește ca date de intrate un form cu 5 câmpuri

- **video**: Videoclipul ce conține imagini cu clasa
- **cls**: Numarul grupei din care fac parte studenții
- **teacher**: Numele profesorului
- **courseName**: Numele cursului
- **courseType**: Tipul cursului, poate fi (SEMINAR, COURSE, LABORATORY)

Rezultat: profesorul și studenții identificați sunt informați de rezultatul procesului

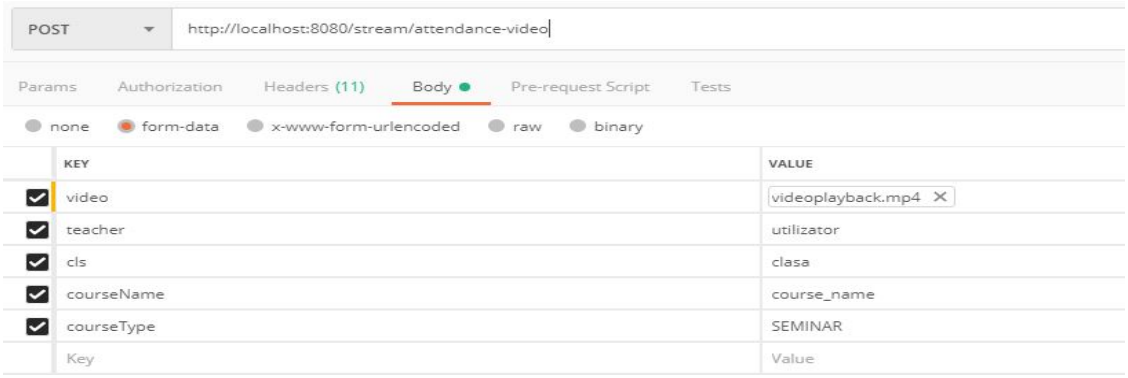


Figure 23: Apelul pentru prezența automată

4.2.8 API pentru a modifica rezultatul prezenței (POST)

Calea de acces(endpoint): /attendance/modify

Acest api este destinat profesorilor, în momentul în care aceștia doresc să modifice rezultatul unui istoric.

Primește ca date de intrare un obiect de tip JSON[**28**] cu cinci câmpuri:

- **historyId**: Id-ul istoricului pe care profesorul dorește să îl modifice
- **type**: Tipul cursului, poate fi (SEMINAR, COURSE, LABORATORY)
- **courseName**: Numele cursului
- **teacherName**: Numele profesorului
- **presents**: O listă cu studentii ce ar trebui să fie prezenți

Rezultat: profesorul și studenții identificați sunt informați de rezultatul procesului

4.3 Aplicația

Pentru o mai bună ilustrare a modului în care este construit sistemul, in subsecțiuniile următoare se vor prezenta diferite diagrame ale aplicației, începând cu **diagrama bazată pe componente**, **diagrama de cazuri de utilizare**, **diagrama de secvență** și **diagrama de clasă**.

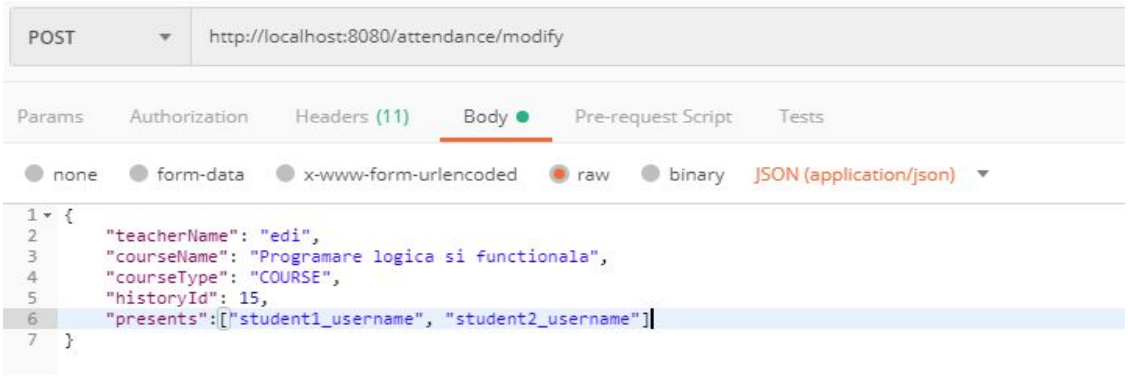


Figure 24: Apelul pentru metoda de modificare a prezenței

4.3.1 Diagrama bazată pe componente

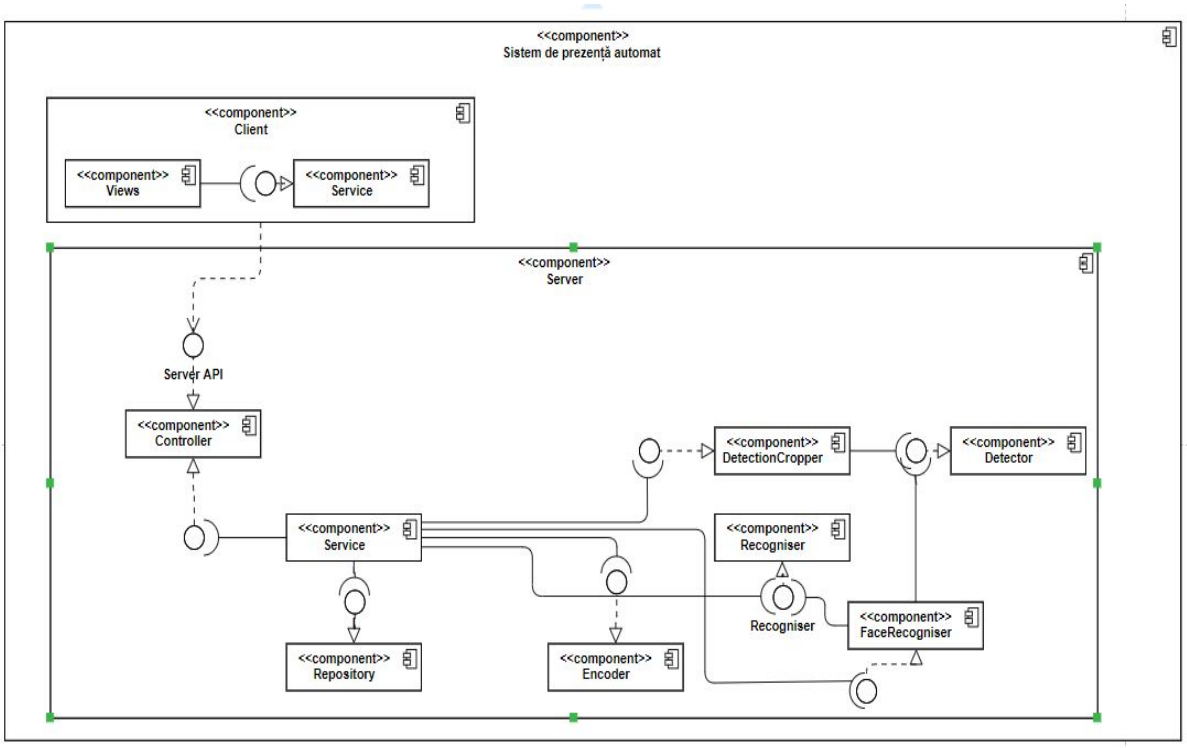


Figure 25: Diagrama bazată pe componente

Diagrama din (fig.25) este folosită pentru a ne arăta relațiile dintre obiecte și modul în care acestea depind unele de altele.

După cum se observă și din diagramă, sistemul este alcatuit din două mari componente: **clientul** și **serverul**. Sistemul se bazează pe sablonul de proiectare client-server, fiind cea mai fezabilă abordare pentru un sistem ce trebuie să funcționeze cum mai mulți utilizatori deodată.

Pe partea de client, lucrurile sunt mai simple, existând două componente principale, view-urile și service-urile. Fiecare view pentru a-și procura datele sau pentru a efectua cererile catre server se folosește de servicii, servicii singleton, din cauza faptului că aceiași instanță a unui service trebuie să fie partajată de către toate view-urile ce îl folosesc.

Pe partea de server, lucrurile devin mai complicate, arhitectura fiind mai complexă. Pentru a nu complica diagrama, am ales să reprezint comonentente de același tip, printr-o singura componentă(controller, service, repository), pentru a economisi spațiu și a nu pierde detaliile esențiale.

Controller-ul este componenta ce îmbină toate celălalte componente de tip controller, scopul acestora fiind crearea de puncte de comunicare între server și client. Acest sistem este alcătuit din urmatoarele tipuri de controller:

- **AuthenticationController**: conține API pentru autentificare, înregistrare, shimbare de parolă, a afla informațiile despre un utilizator, a obține lista de utilizatori și de a seta un anumit rol unui utilizator.
- **HistoryController**: conține API pentru a returna studenții ce au fost sau nu prezenți la o anumită materie, precum și pentru a vedea întreg istoricul de prezențe făcut de un anumit profesor.
- **EnrollmentController**: conține API pentru a adăuga o înscriere a unui student la o materie, pentru a anula o înscriere, pentru a verifica dacă un student este înscris sau nu la o materie, pentru calcula câti studenți sunt înscriși la o anumită materie.
- **ProfileController**: implementează API pentru a salva modificarile asupra profilului unui utilizator, de a afla informații despre profilul acestuia și de a încărca o imagine cu fotografia ce va fi prezentată alături de numele său.
- **StreamingController**: implementeaza API pentru încărcarea de fotografii cu fața utilizatorului cât și a videoclip-ului pentru prezență automată.
- **StudentAttendanceController**: conține API pentru a verifica dacă un utlizator are încărcate imaginile cu fața, pentru a vedea prezențele la o anumită materie (sau la toate materiile), sau pentru modificarea unei prezențe.
- **TeacherCourseController**: conține API pentru a adăuga un nou curs și pentru a vedea toate cursurile postate de un anumit profesor.

Se observă din diagramă că fiecare controller are nevoie de o implementare pentru un service.

Service-ul, aceasta este componenta ce înglobează totalitatea tipurilor de service-uri, acestea implementând logica aplicației. Sistemul este alcătuit din urmatoarele service-uri:

- **AttendanceService**: conține metode necesare pentru toate operațiile ce au legătură cu procesul de prezență.
- **AuthService**: conține metode necesare pentru toate operațiile ce au legătură cu procesul de autentificare.
- **CourseService**: conține metode necesare pentru toate operațiile ce au legătură cu procesul de gestionare al cursurilor
- **ProfileService**: conține metode necesare pentru toate operațiile ce au legătură cu procesul de gestionare al profilului
- **EnrollmentService**: conține metode necesare pentru toate operațiile ce au legătură cu procesul de înscriere la curs
- **RecognitionService**: conține metode necesare pentru toate operațiile ce au legătură cu procesul de recunoaștere facială
- **StreamingService**: conține metode necesare pentru toate operațiile ce au legătură cu procesul de identificare a fețelor din videoclip, pentru a salva imaginile utilizatorului în baza de date
- **HistoryService**: conține metode necesare pentru toate operațiile ce au legătură cu istoricul de prezențe

Fiecare dintre aceste service-uri au nevoie de un repository pentru a putea fi instanțiate, unele chiar și de alte service-uri sau componente auxiliare.

Repository-ul este componenta ce se ocupă de persistență, de interogarea datelor, fiind componenta comună din diagramă pentru urmatoarele repository-uri:

- **AttendanceRepo**: operații CRUD pentru prezențe
- **CourseRepo**: operații CRUD pentru cursuri

- **EnrollmentRepo**: operații CRUD pentru înscriere la cursuri
- **FaceImagesRepo**: operații CRUD pentru imaginile cu fața
- **HistoryRepo**: operații CRUD pentru istoricul de prezențe
- **UserRepo**: operații CRUD pentru utilizator

Pe langă aceste componente mai sunt și componente auxiare folosite in service-uri pentru recunoașterea facială(**FaceRecogniser și Recogniser**), pentru detecția facială (**Detector și DetectorCropper**) și pentru encodarea parolelor (**Encoder**).

4.3.2 Diagrama de cazuri de utilizare

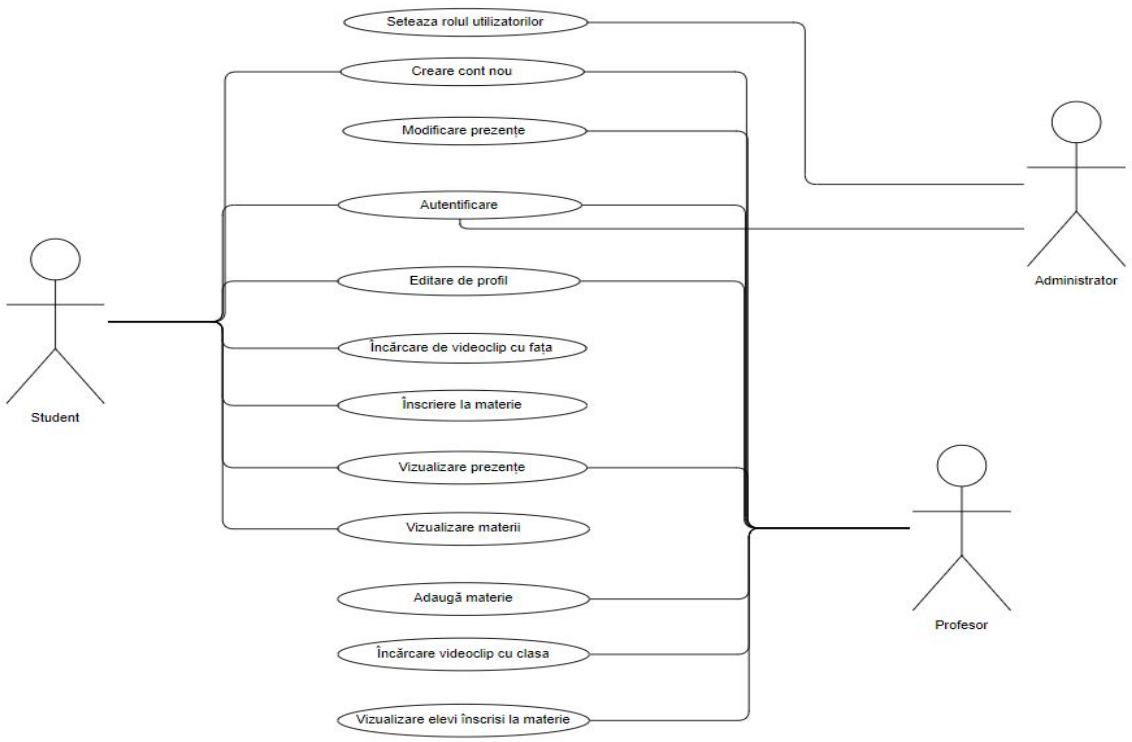


Figure 26: Diagrama de cazuri de utilizare

Această diagramă(fig.26) ne ajută să înțelegem mai bine care sunt actorii și ce acțiuni pot să întreprindă aceștia.

Conform acestei diagrame, identificăm trei actori: **profesorul**, **administratorul**, **studentul**.

Administratorul poate să se autentifice în aplicație și poate seta rolul unui utilizator (fie profesor, fie student).

Studentul poate să își creeze cont, să se autentifice în aplicație, să încarce o fotografie de profil și să editeze profilul, să se înscrie la o materie și să își încarce videoclip cu fața.

Profesorul poate să își creeze cont, să se autentifice în aplicație, să adauge o materie nouă să încarce o poză cu clasa pentru a efectua prezența, să modifice și sa vadă rezultatul prezenței și să încarce o fotografie de profil și să editeze profilul.

4.3.3 Diagrame de secvență

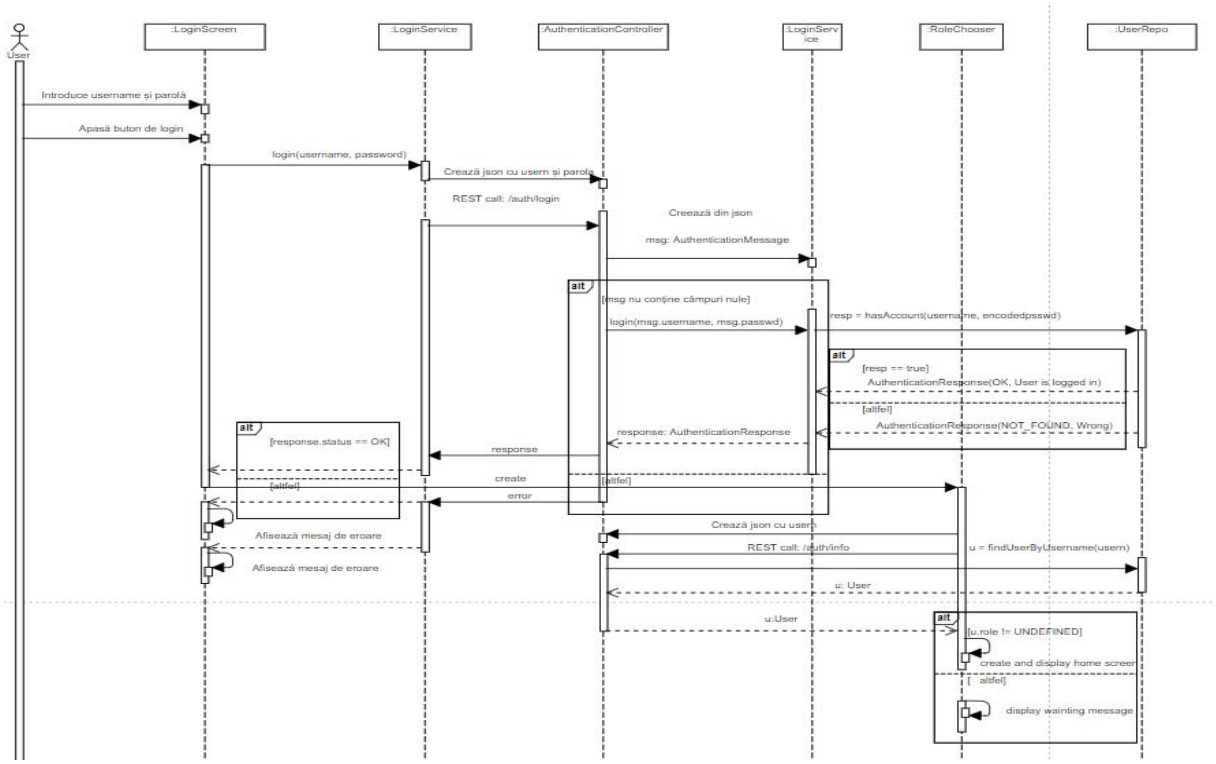


Figure 27: Diagrama de secvență pentru procesul de autentificare

Această diagramă(fig.27) este folosită pentru a ilustra secvența de operații ce se execută pentru procesul de **autentificare**.

Inițial utilizatorul introduce în cadrul view-ului de autentificare, numele de utilizator și parola. Componenta va apela metoda login de pe service-ul de login(LoginService), cu credențialele de autentificare, iar aceasta va face un apel REST către server, pentru a vedea dacă datele introduse sunt corecte.

Controller-ul responsabil pentru acest apel este AuthController. Mai întai, se verifică dacă obiectul primit conține toți parametrii (atât numele de utilizator cât și parola) iar dacă acest criteriu nu este îndeplinit, se va întoarce un mesaj de eroare ce va fi afișat pe view-ul de autentificare.

Dacă obiectul primit are toți parametrii setati corespunzător, atunci se va apela metoda login de pe service-ul de login(AuthService). Aici se criptează parola, și se apelează metoda hasAccount cu numele de utilizator și parola criptată, pentru a vedea dacă în baza de date există un utilizator cu acele credențiale.

În funcție de rezultatul operației se va crea fie un raspuns afirmativ fie unul negativ ce va ajunge înapoi înspre view-ul de login. Dacă răspunsul este negativ, se va afișa un mesaj de eroare iar in caz contrar se va crea un obiect de tipul RoleChooser, responsabil cu afișarea ferestrei portrivite în funcție de rolul utilzatorului.

Se va face un apel REST către același controller și se vor aduce informațiile despre utilizator. În funcție de acestea fie utilizatorul va fi redirectat spre pagina principală corespunzătoare rolului său, fie spre o pagina cu un infinite progress bar.

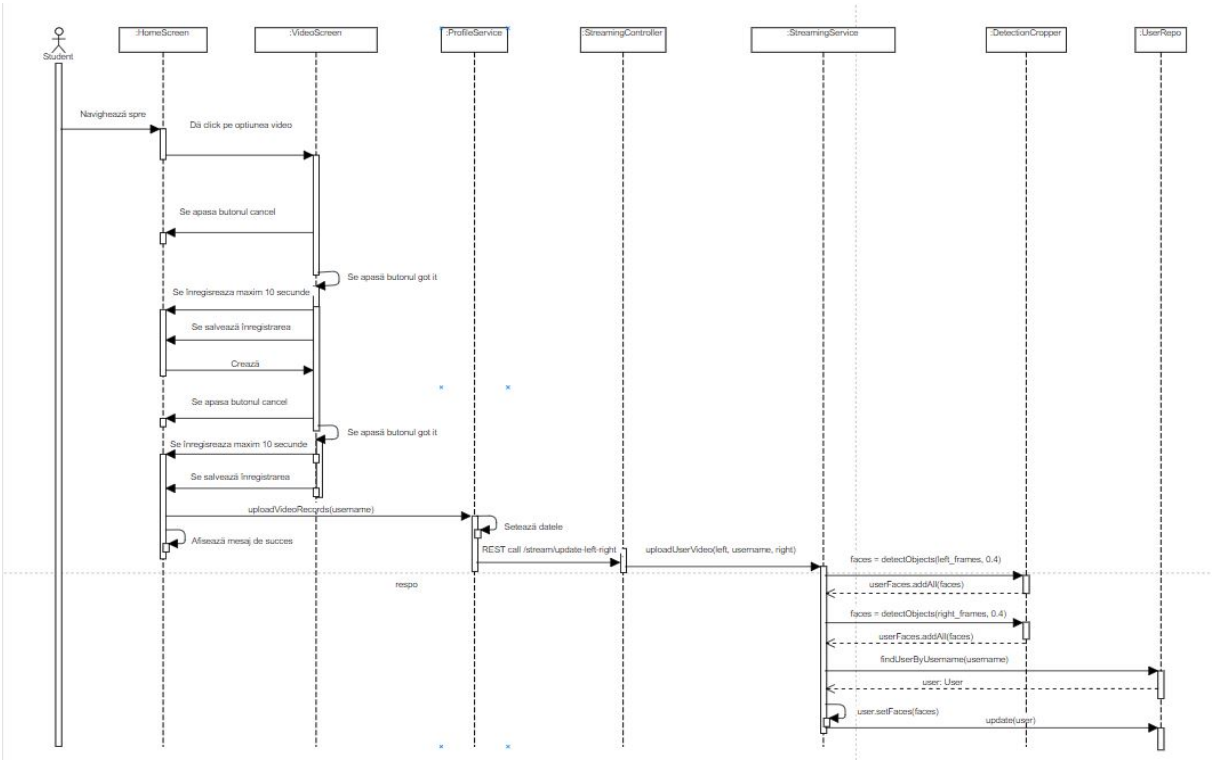


Figure 28: Diagrama de secvență pentru procesul de încărcare a img. cu fața

Această diagramă(fig.28) este folosită pentru a ilustra secvența de operații ce se

execută pentru procesul de **încărcarea a imaginilor cu fața**.

Studentul navighează spre opțiunea de încărcare a videoclip-ului. Acesta poate să aleagă anularea procesului, prin apăsarea butonului de cancel, în acest caz sistemul revenind în starea inițială. Dacă alege să continuie, timp de maxim 10 secunde i se vor înregistra caracterisiticile faciale. După ce au trecut primele 10 secunde, o nouă fereastră va aparea, de această dată pentru înregistrarea mișcărilor orizontale ale capului și feței. Utilizatorul poate oricând să oprească procesul.

După aceea se va apela metoda de upload din service-ul ProfileService și se vor trimite datele către server. Controller-ul StreamingController răspunde la apel delegând sarcina mai departe către service-ul StreamingSerivice. Cu ajutorul componentei DetectionCropper se vor identifica toate imaginile cu fața din cele doua videoclipuri și se vor salva in baza de date în urma apelului update de pe UserRepository.

4.3.4 Diagrama de clasă pentru modulul de recunoaștere facială

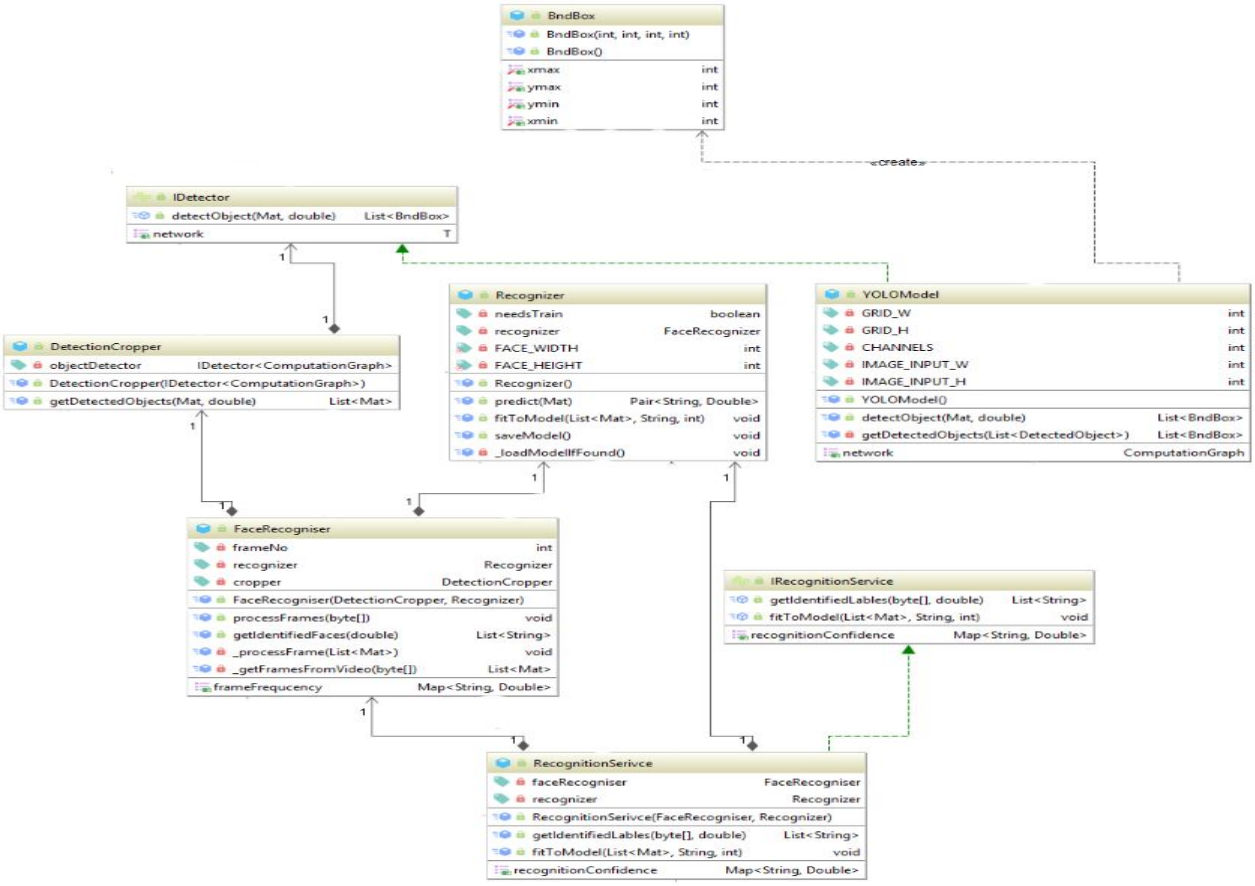


Figure 29: Diagrama de clasă

Această diagramă(fig.29) este folosită pentru a ilustra structura subsistemului de recunoaștere facială.

Componenta principală a acestui subsistem este service-ul de recunoaștere facială (**RecognitionService**).Acesta, implementează interfața IRecognitionService iar pentru realizarea procesului, se folosește de alte două componente, FaceRecogniser, pentru recunoașterea de imagini și Recogniser pentru reantrenarea modelului în caz că se înregistrează utilizatori noi.

FaceRecogniser este componenta ce efectuează procesul de recunoaștere, a persoanelor ce apar într-un videoclip. Pentru detecția fețelor în imagini acesta utilizează o instanță a clasei DetectionCropper, iar pentru a identifica persoanele din imagine, se folosește de o instanță a clasei Regoniser.

DetectionCropper va detecta în poză și va întoarce o listă cu toate fețele ce se regasesc intr-o imagine, folosindu-se de o instanță ce implementează clasa IDetector.

IDetector componentă folosită pentru a detecta zonele din poză ce conțin imagini cu fețe, sub forma unei liste de tupluri cu 4 elemente, fiecare tuplu reprezentând coordonatele colțului stânga sus, dreapta jos al zonei corezpunzătoare.Această interfață are și o implementare concretă, YOLOModel.

YOLOModel componentă pentru localizarea în poză a zonelor ce conțin imagini cu fețe.

4.3.5 Diagrama de bază de date

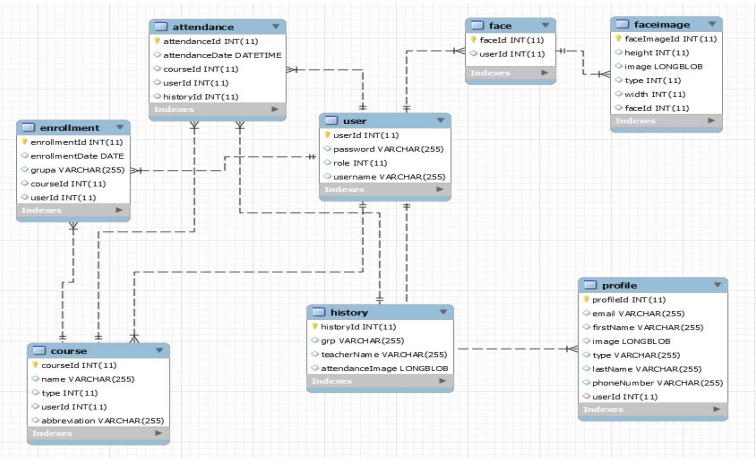


Figure 30: Diagrama de bază de date

În această diagramă(fig. 30) este prezentată structura bazei de date și relațiile dintre entitațile bazei de date. Aceasta a fost modelată astfel încât să se mimeze cât mai bine procesul din viața de zi cu zi. După cum se poate observa din diagramă, baza de date conține 8 tabele: user, face, faceimage, attendance, enrollment, course, history, profile.

- **User**: reprezintă tabelul în care se vor memora informațiile despre utilizator.Acesta poate fi considerat drept tabelul principal, deoarece toate celelalte entitați sunt subordonate acestuia.

- **Face**: acest tabel se afla în legatura one to one cu tabelul user, în incercarea de modelare a procesului natural, fiecare utilizator având o singura față. Poate fi privit ca un tabel intermediar între user și faceimage.

- **FaceImage**: conține informații despre imaginile faciale asociate fiecarui user în parte. Este în legatură many to one cu tabelul face, deoarece unei fețe îi pot corespunde mai multe imagini.

- **Profile**: tabel folosit pentru memorarea informațiilor despre profilul utilizatorului. Este în relație one to one cu tabelul user, fiecărui user corespunzându-i un profil unic.

- **Course**: entitate folosită pentru memorarea tuturor datelor necesare despre un curs. Acesta este într-o relație de tip many to one cu tabelul utilizator cu următoarea seminificație: un utilizator cu rolul de profesor poate să propună mai multe cursuri. Cu tabelul enrollment acesta este în one to many, un curs putând avea mai multe înscrieri iar cu tabelul attendance tot în one to many la un curs putând fi asociate mai multe prezențe.

- **Enrollment**: se memorează informații necesare procesului de înscriere la un curs. Este în relație many to one cu tabelul user, deoarece un user se poate înscrie la o mai multe cursuri.

- **Attendance**: tabel corespuzător înformațiilor legate de prezențe.

- **History**: tabel în care se va memora istoricul prezențelor

4.4 Tehnologii folosite și detalii de implementare

4.4.1 Tehnologii folosite

Pentru implementarea acestui sistem s-au ales diferite tehnologii (JAVA, JAVA Spring, Flutter, Hibernate, MySQL, Deeplearning4j) alături cu diferite medii de implementare (IntellIJ IDEA).

Limbajul de programare JAVA [29]

Acesta este un limbaj de programare de nivel înalt, atât compilat cât și interpretat, orientat obiect, bazat pe clase inventat de James Gosiling și dezvoltat de Oracle Corporation. În ciuda faptului că totul în acest limbaj trebuie scris în interiorul unei clase, acesta nu este pur orientat obiect, conținând și tipuri de date primitive.

Scopul principal pentru care acesta a fost inventat era necesitatea unui limbaj de programare în care codul să fie scris o singura dată și rulat pe cât mai multe sisteme (write once, run anywhere) ce suporta Java fară a mai fi necesară recompilarea.

Aplicatiile Java sunt compilate în așa numitul bytecode, cod ce poate rula pe orice mașină virtuala Java (JVM), indiferent de arhitectura sistemului de calcul. Sintaxa (fig. 31) este asemănătoare cu cea a limbajului C/C++, numai că acesta oferă mai puține facilități low-level. **[29]**

```
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Prints the string to the console.
    }
}
```

Figure 31: Hello word în JAVA [1]

Principalul motiv pentru care această tehnologie a fost aleasă pentru dezvoltarea acestui sistem este pe de o parte necesitatea unui limbaj de programare cu o performanță ridicată apropiată cât mai mult de cea a limbajului C/C++ iar pe de altă parte necesitatea unui limbaj robust, ce permite și are suport pentru diferite librării ce pot fi utilizate pentru a implementarea cât mai rapidă a unei soluții software.

JAVA Spring [30]

Spring este cea mai populara tehnologie pentru dezvoltarea aplicațiilor enterprise în libajul JAVA. Este o platformă open-source, scrisă inițial de Rod Johnson și prima dată publicată cu licența Apache 2.0.

Caracteristicile principale ale framework-ului spring pot fi folosite pentru a dezvolta orice aplicație Java, existând și extensii ale acestei tehnologii ce permit crearea de aplicații web peste platforma JAVA EE. [**30**]

Principalele avantaje ale acestei tehnologii sunt: Dependency Injection și Aspect Orientated Programming. Dependecy Injection (injectarea dependințelor) este un avantaj adus de aceasta platformă, facilitând modul în care obiectele depind unele de altele. Acele depindențe nu sunt nimic altceva decât niște bean-uri (fig. 32) iar injectarea aceastora reprezintă specificarea intr-o componentă a bean-urilor de care aceasta are nevoie (fig. 33).

Figure 32: Specificarea unui Bean

Figure 33: Injectarea unei dependințe

Având în vedere că sistemul este dezvoltat pe șablonul de proiectare client-server, scriere tuturor protocoalelor de comunicare necesare unui serviciu REST nu ar fi fost fezabilă, acestă fiind principalul motiv ce stă la baza alegerii acestei tehnologii pentru implementare.

Flutter [31]

Aceasta este o tehnologie inventata de cei de la Google ce conține mai multe seturi de instrumente pentru crearea aplicațiilor mobile (atât IOS cât și Android), fiind principala metodă pentru crearea aplicațiilor pentru noul sistem de operare, Google Fuchsia.

Engine-ul framework-ului este scris în C++, iar principalul limbaj de programare pentru acest framework este Dart. Având în vedere că acest framework este scris în C++ performanța tuturor aplicațiilor este una foarte bună apropiindu-se foarte mult de cea a coldului nativ scris fie Java pentru android, fie în Objective-C pentru IOS.

```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Welcome to Flutter'),
        ),
        body: Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

Figure 34: Hello word in flutter

Această tehnologie a fost aleasă pentru ușurința prin care se pot dezvolta aplicațiile mobile și pentru obținerea unei performanțe asemănătoare cu cea a codului nativ.

Hibernate ORM [32]

Hibernate este o tehnologie de convertire a datelor între sisteme diferite (ORM), pentru limbajul JAVA. Acesta oferă soluția pentru maparea unui model orientat obiect intr-o bază de date relațională. Este un software gratuit, distribuit sub licența GNU, inventat de Red Hat în anul 2001.

Se folosesc, pentru maparile de date fie fișiere XML fie adnotari JAVA (fig. 35), iar interogările se pot scrie in HQL (hibernate query language).

```
@Id
@JsonIgnore
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int faceImageId;

@Lob
@JsonIgnore
@Column(name="image", columnDefinition="LONGBLOB")
private byte[] image;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "faceId")
private Face face;

@Column
private int height;

@Column
private int width;

@Column
private int type;
```

Figure 35: Exemplu de mapare folosind adnotari

Aceasta tehnologie a fost aleasă pentru facilitarea comunicării cu baza de date, accelerând procesul de dezvoltare al aplicației.

MySQL [33]

MySQL este un sistem de gestiune a bazelor de date relaționale, produs de compania suedeza MySQL AB, fiind cel mai popular SGBD la ora actuală.

Cu toate că este folosit foarte des împreuna cu limbajul de programare PHP, acest framework conține multe API-uri ce îi permit să fie folosit alături de foarte multe limbaje, precum C, C++, C Sharp, Java, pentru accesarea bazelor de date.

Sintaxa acestui limbaj (fig. 36) este foarte asemănătoare cu cea a limbajului SQL, diferind în foarte puține locuri.

```
select enrollments.username, enrollments.absent, enrollments.role from (
    select student.userId as absent, student.username ,student.role from enrollment e
        inner join course c on e.courseId = c.courseId
        inner join user student on e.userId = student.userId
        inner join user teacher on c.userId = teacher.userId
    where
        c.name = 'Programare logica si functionala' and
        c.type = 1 and
        teacher.username = 'edi' and
        e.grupa = '235'
) enrollments where enrollments.absent not in (
    select a.userId as prez from history h
        inner join attendance a on h.historyId = a.historyId
    where h.historyId = 6
)
```

Figure 36: Exemplu de interogare scrisă în MySQL

Aceasta tehnologie reprezintă cea mai bună soluție în momentul în care dorim să lucrăm cu Java Hibernate, datorită existenței unei foarte bune compatibilități intre cele două sisteme, iar acest lucru a fost principa cauza pentru alegerea acestui framework pentru baza de date.

Deeplearning4j [34]

Aceasta este o librărie de învațare profundă, scirsă pentru Java și totodată o librărie de calcul ce conține numeroși algoritmi de învațare profundă, scrisă în Scala, CUDA, C, C++, Python, Clojure.

Datorită faptului că acestă librarie este susținută și dezvoltată de foarte multe instituții reprezintă alegerea perfectă pentru o aplicație Java ce folosește algoritmi de învătare automată.

IntelliJ IDEA [35]

IntelliJ IDEA este o platformă de dezvoltare pentru software dezvoltat de compania JetBrains, lansat în anul 2001. Acesta este deseori folosit pentru facilitatile pe care

le ofera (Inteli Sense) și pentru faptul că suportă o varietate de limbaje de programare.

A fost ales pentru că, după parearea mea, reprezintă cea mai bună alegere pentru scrierea de cod JAVA.

4.4.2 Antrenarea modelului pentru recunoașterea facială

Pentru rezolvarea acestei probleme, am ales să folosesc un algoritm de deep-learning (un model) preantrenat, constrâns fiind de limitele hardware, ce a fost reantrenat cu date „custom" (cu imagini ce conțin fețe), imagini preluate din setul de date WiderFace. Este vorba despre modelul **Tiny YOLO**.

Față de modelul complet YOLO v2, acesta este mai rapid în detecție, căci are mai puține straturi, însă acuratețea acestuia este mai mică decât cea a modelului complet.

Setul de date de antrenament a fost ales pe baza ultimelor descoperiri în domeniu, numeroase lucrări de specialitate făcand referire la aceste imagini când vine vorba de antrenarea modelelor pentru detecție facială, conținând un număr de peste 393000 de poze (fig. 37). Din cauza constrângerilor hardware au fost alese, din acest set de imagini, 2000 de poze.

Figure 37: Imagine preluată din setul de date de antrenament WiderFace

Modelul **YOLO** funcționează în felul următor: imaginea este împărțită în mai multe regiuni, fiecare având rolul de a verifica dacă obiectul ce își are mijlocul în centrul regiunii este sau nu de interes. Totuși, acestui model trebuie să i se spună forma obiectelor, adică, dacă sunt fie mai înalte, fie mai înguste, mai exact, trebuie definite așa numitele anchor-boxes (o pereche de două numere - lațimea și înălțimea unui dreptunghi).

Principala provocare pe care am întâlnit-o, a fost prelucrarea fișierelor de adnotare folosite pentru antrenare, din cauză că biblioteca Deeplearning4j, utilizată pentru inițializarea modelului preantrenat pe imagini, are nevoie ca fișierele de antrenament să fie într-un anumit format, un fomat xml binedefinit, și nu csv (fig. 38).

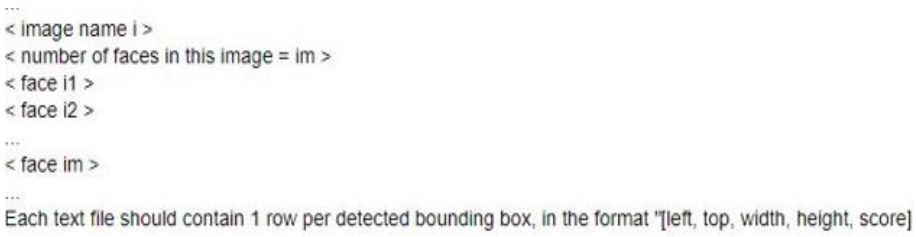


Figure 38: Formatul adnotărilor din setul WiderFace

În acest fișier(fig. 38), datele sunt reprezentate sub formă de cvintete (coordonatele colțului din stânga sus împreună cu lungimea și înălțimea dreptunghiului desenat în jurul zonei din fotografie, în care a fost identificat obiectul căutat).

Pentru antrenare, totuși, avem nevoie de un altfel de format pentru fișierul de adnotare, un format (fig. 39) in care să specificăm pentru fiecare obiect coordonatele colțului din stanga sus dreapta jos (bounding-box-ul ce înconjoara obiectul).

Se poate observa că în acest fisier se specifică pentru fiecare obiect în parte coordonatele dreptunghiului ce înconjoara obiectul.

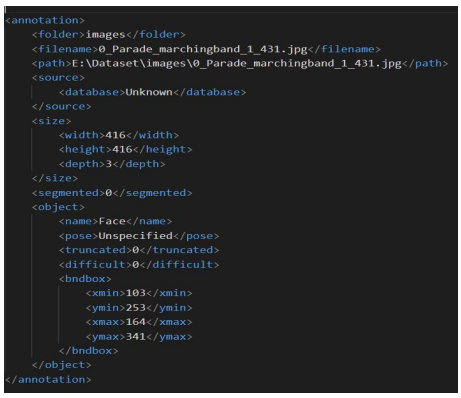


Figure 39: Formatul adnotărilor

4.4.2.1 Algoritmul de creare de fișiere pentru adnotare

Primul pas, pentru a reuși sa antrenez modelul a fost scrierea unui algoritm capabil să extragă din acel fișier csv, informații pentru a construi echivalentul său în format xml, acceptat de modelul antrenat, Tiny YOLO.

În esență, algoritmul este destul de simplu:

1. Se citește o imagine din setul de date de antrenament
2. Se indentifica zona, din fișierul de adnotări original, în care apare numele imaginii curente
3. Citesc numarul N de fețe identificate în această adnotare
4. Cât timp nu am citit cele N linii
5. Citesc cele 4 numere de interes (x, y, w, h) și ignor celelalte informații despre imagine
6. Creez un obiect de tip bndbox (x, y, x+w, y+h) și îl adaug într-un obiect Object
7. Adaug, în lista de obiecte, obiectul creat
8. Repet pasul 1 până când toate imaginile au fost procesate
9. Adaug adnotarea și salvez fișierul, sub numele nume_poza.xml

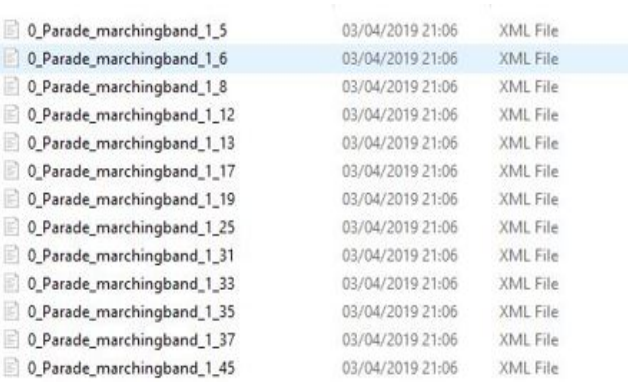


Figure 40: Fisierul obținut

Figure 41: Imaginilie inițiale

După ce acești pași au fost efectuați totul este pregătit pentru antrenarea modelului.

4.4.2.2 Algoritmul de scalare al pozelor și de modificare al bounding-box-urilor

După, a urmat momentul în care am stabilit înălțimea imaginii (416x416) ce intră în primul layer de convoluție, înălțimea unei celule din grid (13), numărul de clase (1), learning rate-ul (.00001) și numărul de epoci (10).

Apoi, am modificat ultimul layer al rețelei (layerul de output) pentru a se potrivi problemei mele (l-am modificat astfel încât să poată să întoarcă acele obiecte (coordonatele) ce au dimenisiuni asemănătoare cu cele ale anchor-box-urilor).

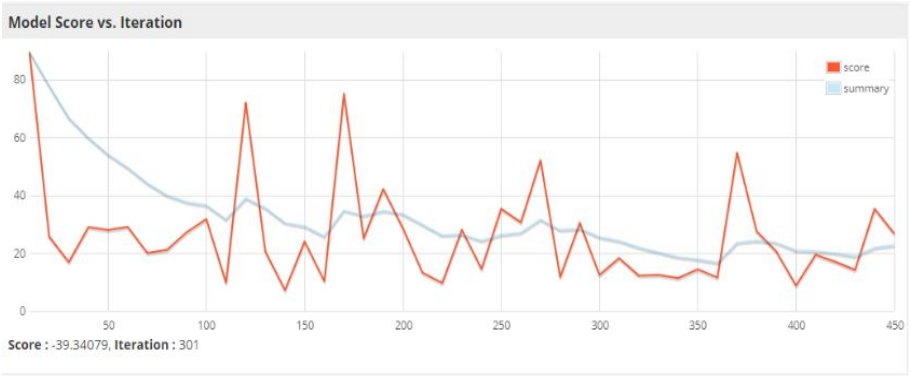


Figure 42: Graficul de loss din timpul antrenării

Se observă că (fig. 42), există perioade în care loss-ul descrește după care crește brusc, ceea ce sugerează că rețeaua nu învață bine iar acest lucru se datorează omiterii a două lucruri importante:

- Dacă imaginea este micșorată la 416x416, atunci și coordonatele bounding-box-urilor ar trebui scalate cu factorul de micșorare al pozei
- Dimensiunea anchor-box-urilor ar trebui setată în funcție de forma obiectelor pe care dorim să le identificăm.

Pentru rezolvarea primei probleme, am implementat un algoritm de scalare al pozelor și a coordonatelor din poză, ce respectă următorii pasi:

1. Parcurg pozele din setul de antrenament
2. Pentru fiecare poză aplic redimensionarea și scalarea coeficienților
    - calculez w = RESIZED_WIDTH / image.width()

- calculez h = RESIZED_HEIGHT / image.height()
- calculez boxHeight = (xmax – xmin)
- calculez boxWidth = (ymax – ymin)
- x1 = w * xmin
- x2 = h * ymin
- x2 = x1 + w * boxHeight
- y2 = y1 + h * boxWidth

3. Rescrierea fișierul xml de adnotare corespunzător fiecărei imagini

După ce am rulat, cu noile date, rezultatele au fost mai bune dar nu satifăcătoare, iar după ajustarea parametrilor ce pot fi configurați (alegerea Adam ca optimizator pentru evitarea exploding-gradients (fig.43) și vanish-gradient problem), schimbarea learning rate-ului, am realizat că există un detaliu care îmi scapă și anume identificarea dimensiunii corecte a anchor-box-urilor.

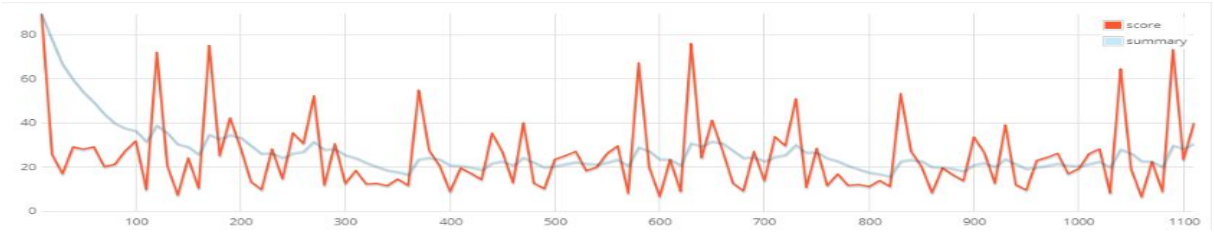


Figure 43: Exploding gradients

4.4.2.3 Algoritmul K-Means folosit pentru determinarea anchor-box-urilor

După o documentare amănunțită, am realizat că nu pot obține rezultate satisfăcătoare cu valorile default pentru anchor-boxes, așa că, trebuia implementat un algoritm pentru determinarea anchor-box-urilor, pe setul actual de imagini.

Ideea pe care am folosit-o, a fost aceea de a folosi un algoritm de clusterizare pentru a împărți datele în BOX_NUMBER clusteri, iar centroidul fiecărui cluster să reprezinte dimensiunile anchor-box-ului, un algoritm de clusterizare K-Means, inspirat din [**36**].

Însă pentru aceasta trebuia cumva ca bounding-box-urile să fie transformate în puncte, P(x,y), relativ la problema pe care dorim să o rezolvăm.Punctele au fost obtinute după această formulă:

$$x = \frac{(x_{max} - x_{min})}{width} * 32 \tag{7}$$

$$y = \frac{(y_{max} - y_{min})}{height} * 32 \tag{8}$$

- x_{max} și x_{min}: reprezintă abscisele coordonatelor bnd-boxurilor
- y_{max} și y_{min}: reprezintă ordonatele coordonatelor bnd-boxurilor
- height: reprezintă înalțimea imaginii
- width: reprezintă latimea imaginii
- 32 reprezintă numărul de transformări pe care le suferă in rețea datele (acestea fiind micșorate de 32 de ori în cadrul rețelei)

După rularea algoritmului de cluserizare de date, vom lua drept anchor-boxes valorile centroizilor (prima coordonată reprezentând lătimea iar cea de a doua înălțimea), adică acele puncte ce au abscisa egală cu media tututor absciselor iar ordonata egală media tuturor ordonatelor din acel cluster.

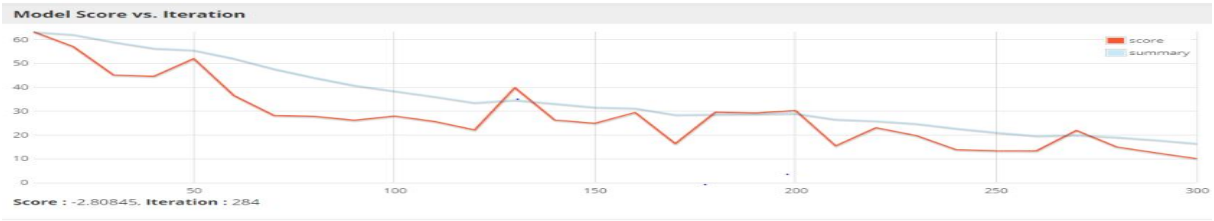


Figure 44: Exploding gradients

Se observă în(fig. 44) cum rețeaua reușește să învețe, loss-ul scăzând liniar, cu foarte puține variațiuni. După antrenare, modelul obținut este salvat intr-un fișier cu numele **model.data**

4.4.3 Încărcarea modelului și integrarea acestuia în sistem

Modelul salvat anterior este integrat intr-o componentă ce pe baza coordonatelor întoarse de acesta, va extrage din imaginea initială zonele cu fețe, salvandu-le într-o listă.

```java
public static ComputationGraph getModel() throws Exception {

    final String modelName = ConstantsManager.getInstance().get("modelPath");

    File file = new File(modelName);

    if (!file.exists() || file.isDirectory()) {
        throw new Exception("Saved detection_model not found!");
    }

    return ModelSerializer.restoreComputationGraph(file);
}
```

Figure 45: Încărcarea modelului in componentă

4.4.4 Implementarea sistemului

După ce au fost implementate toate componentele necesare procesului de recunoaștere facială s-au dezvoltat, pe rând atât API-urile prin care clientul poate comunica cu serverul cât și interfața pentru client.

Pentru a spori acuratețea detecția de prezență automată, s-a luat decizia ca pentru fiecare student în parte să se memoreze o listă de imagini(ce conțin fața), obținută prin încărcarea a două videoclipuri (unul în care mișcările capului sunt pe verticală iar celălalt pe orizontală).

Cu ajutorul componentei menționate mai sus se obtine lista tuturor fețelor identificate in acel videoclip (se aplica detecția pe fiecare cadru al videoclipului în parte).

Pentru a memora numai cadre relevante pentru algoritmul de recunoaștere facială, s-a luat decizia ca algoritmul de detecție facială să fie din K în K (fig.46) cadre, obținandu-se astfel imagini cât mai diversificate.

```java
// jump over FRAMES_PER_SEC / REDUCTION_RATE frames from each step
for (int i = 0; i < frames.size(); i += (FRAMES_PER_SEC / REDUCTION_RATE)) {

    final opencv_core.Mat frame = frames.get(i);
    final List<Mat> faces = cropper.getDetectedObjects(frame,  detection: .4);

    //debug
    __saveDetectionIntoFileDEBUG(i, faces, isLeftRight);

    userFaces.addAll(faces);
}
```

Figure 46: Eliminarea cadrelor asemănătoare

Vom reantrena algoritmul de recunoaștere facială (LBPH), cu noile imagini obținute cu fața utilizatorului, iar pentru fiecare imagine, vom folosi drept indentificator ID-ul utilizatorului.

```
//retrain the model to accept the new user face
recognitionService.fitToModel(
        userFaces, user.getUsername(), user.getUserId()
);
```

Figure 47: Reantrenarea modelului de recunoaștere facială

4.4.4.1 Algoritmul de prezență automată

Pentru efectuarea automată a prezențelor se folosesc ambele componente: componenta de recunoaștere facială și componenta de detecție facilă. Profesorul va încărca un videoclip de maxim 10 secunde al clasei, iar acesta va fi descompus în cadre. Pașii algoritmului de prezență automată sunt:

1. Se alege cadrul curent din lista de cadre disponibilă
2. Se identifică zonele din poză in care sunt fețe și se extrag
3. Se aplică algoritmul de recunoaștere facială pentru a i se asocia fiecărei poze câte o etichetă
4. Se contorizează pentru fiecare etichetă numarul de apariții
5. Se trece la cadrul următor și se repetă pasul 1, dacă mai sunt cadre în listă.
6. Se numară în câte dintre toate aceste cadre s-a detectat cel puțin o față
7. Se împarte numarul de apariții al fiecărei etichete la numarul obținut anterior și se memorează rezultatul obținut
8. Dacă procentajul de apariție este mai mare sau egal cu valoarePrag atunci se adaugă prezența și utilizatorul (studentul) va fi notificat

Figure 48: Prezența automată

5 Rezultate

5.1 Descrierea setului de date de test și configurația sistemului

Testul a fost executat pe un computer cu un procesor Intel Core i7-8750H de 4,1Ghz, memorie RAM de 16Gb DDR4 și un GPU Nvidia GeForce GTX 1070 SC de 8Gb VRAM.

Acesta constă în antrenarea rețelei neuronale pe fiecare dintre cele 3 seturi de date și rularea testelor pe fiecare în parte. S-a folosit o proporție de 80-20 pentru datele de antrenare respectiv test.

Seturile de date de antrenament sunt alcătuite din 2000 de imagini in care se regasesc oameni în diferite poziții. Pentru primul test s-au ales 2000 de imagini din setul de date WIDER Face(Easy) [**37**], pentru cel de al doilea, 2000 de imagini din setul WIDER Face(Medium) [**37**] iar pentru ultimul 2000 de imagini din WIDER Face(Hard) [**37**].

Dificultatea acestor teste este datorată colplexității imaginilor pe care rețeaua le folosește pentru a învăța.

În setul de date Wider Face Easy imaginile cu fețele oamenilor sunt clare și la o distanță nu prea îndepărtată pe când în setul Medium și Hard acestea sunt fie mai neclare, fie foarte îndepartate, fie incomplete, rețeaua trebuind să învețe atât texturi cât și caracteristici faciale.

Figure 49: WIDER Easy

Figure 50: WIDER Medium

Figure 51: WIDER Hard

5.2 Rezultatele obținute

Pentru a măsura perfomanța modelului de detecție s-au folosit urmatoarele metrici:

True Positives(TP): reprezintă numărul acelor bounding-box-uri ce au IoU (Intersection over Union) mai mare de 0.5

False Positives(FP): reprezintă numărul acelor bounding-box-uri ce au IoU (Intersection over Union) mai mic de 0.5 sau aceleora ce au mai fost detectate

True Negatives(TN): în cazul acesta, această valoare este 0 deoarece imaginea trebuie să conțină cel puțin un obiect

False Negatives(FN): reprezintă numărul acelor imagini ce nu au produs bounding-box-uri.

- Precision: măsoară rația dintre instanțele relavante în raport cu instantele preluate

$$\frac{TP}{TP + FP}$$

- Accuracy: măsoară procentajul de instanțe identificate corect de model

$$\frac{TP + TN}{TP + TN + FN + FP}$$

- Recall: masoară numărul de instanțe pozitive ce au fost identificate corect

$$\frac{TP}{TP + FN}$$

DataSet	Accuracy	Precision	Recall
WIDER Face(Easy)	70.35%	72.5%	71.8%
WIDER Face(Medium)	52.12%	65.52%	70.42%
WIDER Face(Hard)	45.39%	62.35%	63.18%

Se poate observa din acest tabel că rezultatele obținute nu sunt foarte satisfăcătoare, cea mai mare valoare pentru acuratețe fiind 70.35%. În practică însă s-a constatat că algorimul funcționează destul de bine, reusind să detecteze obiecte la distanțe acceptabile.

Pe de o parte acest rezultat negativ, în opinia mea este datorat numărului mic de imagini necesar pentru antrenare și al constrângerilor hardware iar pe de altă parte, a datelor din acest set de antrenament.

Faptul că imaginile nu au aceiași dimensiune reprezintă o problemă pentru procesul de învățare, problemă ce a fost rezolvată prin redimensionarea acestora, insă acest proces de redimensionare poate avea impact asupra consistenței datelor, deoarece pot apărea distorsionări ale obiectelor.

Așa cum se observă și din ultima înregistrare a tabelului se pare ca modelul nu se „descurcă bine", pe imagini ce conțin obiecte foarte îndepărtate, dar acest lucru era de așteptat, deoarece prin folosirea unei rețele neuronale cu un număr mic de straturi(TinyYOLO) se sporește performanța și nu acuratețea.

6 Probleme întâlnite

Prima problemă cu care m-am confrutat a fost accea de a găsi date de antrenament, adnotate, pentru antrenarea rețelei neuronale, deoarece majoritatea seturilor de date pe care le găseam fie nu erau adnotate, fie erau adnotate dar formatul adnotării nu era unul corespunzător problemei de detecție de obiecte.

După găsirea setului de date de antrenament, urmatoarea problemă a fost alegerea unei biblioteci cu ajutorul căreia să pot antrena un model de recunoaștere de obiecte, nu de la zero, ci un model preantrenat, prentru libajul Java. După o documentație amănunțită, am ales libraria Deepleaning4j.

Un dezavantaj al acestei librarii a fost acela că nu era specificat formatul în care ar trebui să fie adnotările imaginilor, in care sunt specificate chenarele (bounding-box-urile) din jurul fiecărei fețe din poză.

După ce am aflat formatul datelor, următoarea problemă a fost acceea că formatul acestora era diferit de cel oferit în datele de antrenament, iar pentru a rezolva acest lucru a trebuit ca fiecare adnotare să fie rescrisă în formatul acceptat de modelul de antrenare.

Următoarea problemă întâlnită, a fost alegerea tehnologiei potrivite pentru dezvoltarea aplicației de mobil, ce să permită dezvoltarea rapidă și să ofere o bună performanță și alegerea unei arhitecturi pentru aplicatie (atât pentru server cât și pentru clientul mobil) ce să permită adaugarea funcționalităților noi cu un număr minim de modificări.

7 Concluzii și direcții viitoare

7.1 Concluzii

Prezența la ore are un rol foarte important în obținerea unei bune performanțe academice iar datorită acestui fapt aceste instituții doresc, pe cât de mult se poate, să identifice absenteismul studentilor și să diminueze numărul de studenți ce nu sunt prezenți la ore.

Deși există metode clasice de efectuare a prezențelor, acestea s-au dovedit a fi depașite căci, traind în epoca internetului și a informației, totul este se axează pe o cât mai mare viteză, încercandu-se minimizarea timpului necesar efectuarii oricarei sarcini, indiferent de domeniul de provenientă.

Deși există numeroase abordări pentru această problemă, acestea s-au dovedit a fi ori prea scumpe, ori incomplete din punct de vedere al funcționalităților oferite, fie prea lente, necesitând zile pentru efectuarea în mod automat al prezenței.

În această lucrare se propune o soluție automată pentru detecția și realizarea automată a prezențelor, bazată pe tehnici de machine learning și computer vison, ce folosește recunoașterea facială pentru identificarea studenților, lasând totuși controlul profesorului în finalizarea rezulatelor, ce poate fi folosită cu succes in orice instituție academica.

7.2 Direcții viitoare

Soluția propusă este fezabila, comportandu-se bine în practică, dar cu toții știm că nimic nu poate fi perfect, mereu existând loc pentru îmbunatățiri. Acest sistem cu toate că se descurcă bine într-un mediu real, poate fi modificat astfel încât să obțină rezultate și mai bune.

Ar putea fi adaugată o funcționalitate prin care la un anumit interval de timp, fișierul în care este salvat modelul pentru recunoaștere facială să fie șters, pentru a forța algoritmul să se antreneze numai cu fețele actuale ale utilizatorilor, eliminând vechile înregistrări. Acest proces pe lângă faptul că ar face ca modelul să fie mai rapid în detecție (căci dimensiuea acestuia va scadea) ar spori și acuratețea, menținandu-se numai date actuale.

Modulul de perisistența a datelor ar putea fi reimplementat astfel încât să se sporească viteza cu care datele sunt accesate din baza de date, adică toate operațiile pe baza de

date ar trebui scrise in HQL, evitând folosirea metodelor deja existente.

Adăugarea unei funcționalități prin care profesorul să poată genera rapoarte în format excel referitoare la istoricul prezențelor, statistici legate de prezența sau adaugarea unui subsitem pentru a le aminti studenților ziua și ora la care aceștia ar trebui să aiba o anumită materie (fie ca e curs, seminar sau laborator).

Exstinderea aplicației cu un client web, pentru a putea vedea în timp real acele statitici/rapoarte adăugate. După autentificare, profesorul poate să creeze dinamic rapoarte folosind buissness inteligence, rapoarte ce se actualizează în timp real, care ar putea oferi soluții pentru îmbunătățirea numarului de prezențe al studenților la ore.

Folosirea altui model pentru recunoașterea facială. Având în vedere că acest model nu obține o acuratețe atât de bună, comparativ cu cel mai bun detector de obiecte în timp real existent, s-ar putea folosi acel model, sau un API pentru o mai bună detecție facială.

O existindere al acestui sistem ar putea fi folosirea de camere, pentru prezența automată. Asfel profesorul ar putea să precizeze sala, iar sistemul să detecteze în mod automat grupa ce ar trebui să aiba oră in acea clasă. În acest mod, profesorul nu ar trebui să se mai implice în procesul de prezență automată.

Informarea utilizatorului, în mod regulat, deprese situația prezențelor acestuia, prin intermediul poștei electronice (email).

Referințe bibliografice

[1] Java (programming language), Jun 2019.

[2] Visar Shehu and Agni Dika. Using real time computer vision algorithms in automatic attendance management systems. In *Proceedings of the ITI 2010, 32nd International Conference on Information Technology Interfaces*, pages 397–402. IEEE, 2010.

[3] Yohei Ishii, Hitoshi Hongo, Kazuhiko Yamamoto, and Yoshinori Niwa. Face and head detection for a real-time surveillance system. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 3, pages 298–301. IEEE, 2004.

[4] Abhishek Jha. Class room attendance system using facial recognition system. *The International Journal of Mathematics, Science, Technology and Management*, 2(3):4–7, 2007.

[5] K Susheel Kumar, Vijay Bhaskar Semwal, and Ramesh Chandra Tripathi. Real time face recognition using adaboost improved fast pca algorithm. *arXiv preprint arXiv:1108.1353*, 2011.

[6] Paul Viola, Michael Jones, et al. Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, 1:511–518, 2001.

[7] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (pca). *Computers & Geosciences*, 19(3):303–342, 1993.

[8] Vishal Bhalla, Tapodhan Singla, Ankit Gahlot, and Vijay Gupta. Bluetooth based attendance management system. *International Journal of Innovations in Engineering and Technology (IJIET) Vol*, 3(1):227–233, 2013.

[9] Barroon Ismaeel Ahmad et al. Touchin: An nfc supported attendance system in a university environment. *International Journal of Information and Education Technology*, 4(5):448, 2014.

[10] Bill Joy, Guy Steele, James Gosling, and Gilad Bracha. The java language specification, 2000.

[11] Seifedine Kadry and Mohamad Smaili. Wireless attendance management system based on iris recognition. *Scientific Research and essays*, 5(12):1428–1435, 2013.

[12] Basheer KP Mohamed and CV Raghu. Fingerprint attendance system for classroom needs. In *2012 Annual IEEE India Conference (INDICON)*, pages 433–438. IEEE, 2012.

[13] Unnati A Patel and S Priya. Development of a student attendance management system using rfid and face recognition: A review. *International Journal of Advance Research in Computer Science and Management Studies*, 2(8):109–119, 2014.

[14] What is machine learning a definition, Oct 2017.

[15] Jacek M Zurada. *Introduction to artificial neural systems*, volume 8. West publishing company St. Paul, 1992.

[16] Ravindra Parmar and Ravindra Parmar. Common loss functions in machine learning, Sep 2018.

[17] Avinash Sharma V and Avinash Sharma V. Understanding activation functions in neural networks, Mar 2017.

[18] Muhammad Rizwan. Rmsprop, May 2018.

[19] Rohith Gandhi and Rohith Gandhi. A look at gradient descent and rmsprop optimizers, Jun 2018.

[20] Kazuya Kakizaki and Kosuke Yoshida. Adversarial image translation: Unrestricted adversarial examples in face recognition systems. *arXiv preprint arXiv:1905.03421*, 2019.

[21] How convolutional neuronal network works?

[22] Amar Budhiraja and Amar Budhiraja. Dropout in (deep) machine learning, Dec 2016.

[23] Jonathan Hui and Jonathan Hui. map (mean average precision) for object detection, Mar 2018.

[24] Matti Pietikäinen. Local binary patterns.

[25] Kelvin Salton, Prado, Kelvin Salton, and Prado. Face recognition: Understanding lbph algorithm, Nov 2017.

[26] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *international Conference on computer vision & Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE Computer Society, 2005.

[27] Daniel J Strom and Ohad Zeliger. Methods and apparatus providing remote operation of an application programming interface, March 7 2006. US Patent 7,010,796.

[28] Tim Bray. The javascript object notation (json) data interchange format. Technical report, 2017.

[29] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[30] Tutorialspoint.com. Spring framework overview.

[31] Flutter - beautiful native apps in record time.

[32] Elizabeth J O'Neil. Object/relational mapping 2008: hibernate and the entity data model (edm). In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1351–1356. ACM, 2008.

[33] Paul DuBois and Michael Foreword By-Widenius. *MySQL*. New riders publishing, 1999.

[34] D Team et al. Deeplearning4j: Open-source distributed deep learning for the jvm. *Apache Software Foundation License*, 2, 2016.

[35] Intellij idea: The java ide for professional developers by jetbrains.

[36] K-means clustering for anchor boxes, Apr 2018.

[37] Shuo Yang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5525–5533, 2016.