

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

**Identificarea similarității vizuale a imaginilor
folosind rețele neurale**

propusă de

Petru-Marian Gafițescu

Sesiunea: iulie, 2019

Coordonator științific

Conf. Dr. Sabin-Corneliu Buraga

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ

**Identificarea similarității vizuale a
imaginilor folosind rețele neurale**

Petru-Marian Gafițescu

Sesiunea: iulie, 2019

Coordonator științific

Conf. Dr. Sabin-Corneliu Buraga

Avizat,
Îndrumător lucrare de licență,
Conf. Dr. Sabin-Corneliu Buraga.

Data: Semnătura:

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Gafițescu Petru-Marian** domiciliat în **România, jud. Suceava, Sat. Arbore(Com. Arbore), nr. 1012**, născut la data de **30 iunie 1998**, identificat prin **CNP 1980630330527**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2019, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Identificarea similarității vizuale a imaginilor folosind rețele neuronale** elaborată sub îndrumarea domnului **Conf. Dr. Sabin-Corneliu Buraga**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data:

Semnătura:

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Identificarea similarității vizuale a imaginilor folosind rețele neurale**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Petru-Marian Gafițescu**

Data:

Semnătura:

Cuprins

| | |
|---|-----------|
| Rezumat | 2 |
| Introducere | 3 |
| 1 Fundamente | 5 |
| 1.1 Rețea neurală | 5 |
| 1.2 Rețele neurale convoluționale | 9 |
| 1.2.1 Convoluția | 10 |
| 1.2.2 <i>Pooling</i> | 10 |
| 1.2.3 <i>Clasificare</i> | 11 |
| 1.3 Keras și Tensorflow | 11 |
| 1.3.1 Tensorflow | 11 |
| 1.3.2 Keras | 12 |
| 1.4 OAuth | 12 |
| 1.5 Angular | 13 |
| 1.6 Flask | 14 |
| 2 Scopuri și cerințe | 15 |
| 2.1 Necesitatea aplicației | 15 |
| 2.2 Modul în care aplicația rezolvă problema vizată | 15 |
| 2.3 Comparația cu soluții existente | 16 |
| 2.3.1 SimilarityPivot | 16 |
| 2.3.2 Reverse image search | 16 |
| 2.3.3 Servicii pentru analiza imaginilor | 17 |
| 3 Analiză și proiectare | 18 |
| 3.1 Scenarii de utilizare | 18 |
| 3.1.1 Logare | 18 |

| | | |
|----------|---|-----------|
| 3.1.2 | Alege sursa imaginilor | 18 |
| 3.1.3 | Încarcă imagine | 18 |
| 3.2 | Arhitectura aplicației | 19 |
| 3.3 | Planificarea activităților realizate de către aplicație | 21 |
| 3.4 | Modulul <i>back-end</i> | 22 |
| 3.4.1 | API-ul expus | 22 |
| 3.4.2 | Rețeaua neurală folosită | 23 |
| 3.4.3 | Baza de date | 26 |
| 4 | Detalii de implementare | 27 |
| | Concluzii | 28 |
| | Bibliografie | 29 |
| | Anexe | 31 |

Rezumat

Această lucrare descrie scopul, implementarea și rezultatele aplicației web "FlickrCloneFetch", o aplicație web ce permite utilizatorului să se conecteze cu ajutorului contului de *Flickr*, să încarce în aplicație o imagine, iar apoi să vizualizeze imaginile similare din punct de vedere vizual. În funcție de preferințele utilizatorului, imaginile vor fi preluate de pe profilul său, imaginile publice ale contactelor sale sau imaginile publice relevante de pe platforma *Flickr*.

Modulul de *back-end* al aplicației este scris în Python, folosind Flask pentru a expune un API. Analiza imaginilor este făcută cu ajutorul unui model de rețea neurală convoluțională, folosind *framework*-urile Tensorflow și Keras. *Front-end*-ul aplicației este realizat în Typescript, utilizând platforma Angular. Sursa imaginilor este platforma *Flickr*, imaginile fiind accesate folosind *Flickr API*.

Introducere

În zilele noastre, datorită faptului că majoritatea telefoanelor mobile au și funcția de cameră foto, imaginile au devenit principala formă de rememorare a evenimentelor trăite. Platforme întregi sunt dedicate fotografiilor, printre cele mai populare numărându-se Instagram, Flickr sau Pinterest. Multe din aceste platforme oferă posibilitatea de sortare, respectiv căutare a imaginilor în funcție de locație, mărime, data fotografiei sau descrierea adăugată de utilizator. Uneori aceste opțiuni nu sunt de ajuns, imaginile având mai multă relevanță din prisma conținutului decât după orice alte metadate. Totuși, opțiunea de a căuta imagini după similaritatea vizuală față de altă imagine, cu alte cuvinte, chiar după componența imaginii este o caracteristică rar întâlnită.

Aplicația "FlickrCloneFetch" își propune implementarea acestei caracteristici pentru platforma *Flickr*, oferind utilizatorului posibilitatea de a vedea imagini similare vizual cu o imagine încărcată atât de pe propriul profil, cât și de pe profilele contactelor sau chiar din imaginile publice de pe platformă.

Scopul "FlickrCloneFetch" este de a fi o unealtă utilă atât fotografilor care folosesc platforma (pentru a vedea lucrările din propriul profil care au aceeași tematică), cât și utilizatorului uzual (pentru a găsi fotografii legate de un anumit eveniment din viața sa, sau fotografii cu unele tematici alese).

Pentru realizarea funcționalității descrise au fost folosite diverse tehnologii web, precum Flak, Angular, împreună cu API-ul platformei *Flickr* și un model de rețea neurală specializată în clasificarea imaginilor, model disponibil prin *framework*-ul TensorFlow.

Capitolul *Fundamente* va oferi o descriere a principalelor concepte, tehnologii și biblioteci folosite în implementarea aplicației.

Scopul urmărit, o privire de ansamblu asupra aplicației și comparația cu unele soluții deja existente se vor regăsi în capitolul *Scopuri și cerințe*.

Următorul capitol, *Analiză și proiectare* conține descrierea arhitecturii aplicației, prezentarea componentelor necesare și a modului în care interacționează, precum și scenariile de utilizare ale aplicației.

În capitolul *Detalii de implementare* se prezintă aspecte privitoare la modul de implementare, motivarea alegerii modelului de rețea neurală, algoritmi și structurile de date folosite pentru a optimiza timpul de răspuns al aplicației precum și API-urile externe folosite.

Capitolul *Manual de utilizare* prezintă modul de utilizare a aplicației , fiind urmat de *Concluzii*, o serie de concluzii legate de modul în care aplicația răspunde cerințelor și posibile direcții de dezvoltare ulterioară.

Capitolul 1

Fundamente

1.1 Rețea neurală

O rețea neurală este un ansamblu interconectat de elemente de procesare simple, *unități* sau *noduri*, a căror funcționalitate este ușor bazată pe neuronul biologic. Abilitatea de procesare a rețelei este stocată în conexiunile inter-unități, sau *ponderi* sinaptice, obținute dintr-un proces de adaptare la învățare dintr-un set de date de antrenare.[7]

În biologie, fiecare neuron primește informație prin dendrite și transmite informație prin axon, doar atunci când a primit suficientă informație de la neuronii conectați la dendritele sale. Astfel, un neuron va transmite informația mai departe atunci când cantitatea de informație primită depășește un anumit *prag*.

Neuronii artificiali sunt construiți pe același principiu: dendritele vor fi echivalate de conexiuni către unitate, iar axonul de către o conexiune pornind de la acea unitate. Fiecare sinapsă(conexiune inter-neuronală) va fi caracterizată de o anumită *pondere* astfel încât datele de intrare primite prin acea conexiune vor fi multiplicare cu acea pondere înainte de a fi transmise la neuronul artificial. Aici, semnalele ponderate sunt însumate pentru a forma o *activare*.

În cadrul nodului, dacă *activarea* va depăși un anumit *prag*, unitatea va transmite mai departe o valoare(de obicei 1), altfel nu transmite nimic. Compararea cu pragul de activare se face folosind o funcție de activare. Matematic, pentru datele de intrare reprezentate de $x_1...x_n$ și ponderile reprezentate de $w_1...w_n$, neuronul artificial funcționează astfel:

$$output = \begin{cases} 0, & \text{dacă } \sum_i x_i * w_i \leq prag \\ 1, & \text{dacă } \sum_i x_i * w_i > prag \end{cases}$$

Procedeul descris (însumare, compararea cu pragul, apoi emiterea de 1 când pragul este mai mic și 0 altfel) reprezintă cel mai simplu model de neuron artificial, perceptronul (fig. 1.1).

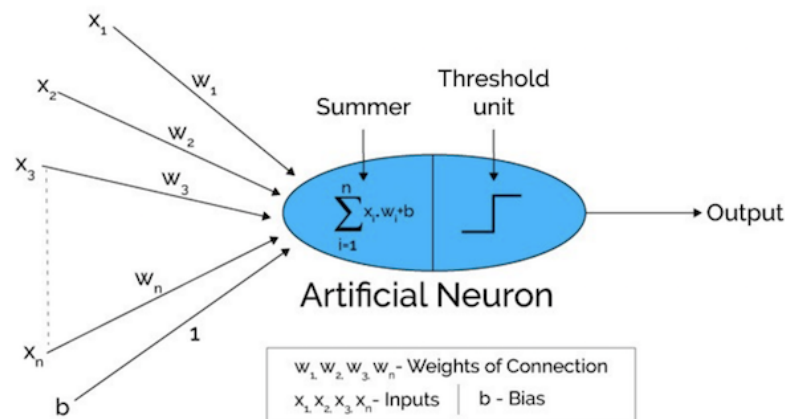


Figura 1.1: Un model de perceptron[5]

Modul de învățare a unui perceptron este foarte simplu: Dacă greșește în a clasifica un element (obține *output* 1 când valoarea elementului e 0, sau invers) creștem ponderile dacă activarea a fost prea mică comparativ cu pragul de activare, respectiv descreștem ponderile dacă pragul de activare a fost mai mic decât activarea. Astfel perceptronul își adaptează ponderile dinamic, în funcție de datele de antrenament, trecând printr-un proces de învățare.

De-a lungul istoriei, au fost folosite mai multe tipuri de neuroni, fiecare având avantajele și dezavantajele sale. Cum corpul neuronului este în principiu un sumator ponderat, ce se va schimba între tipurile de neuroni este funcția de activare.

Deoarece de unul singur un neuron nu are putere de decizie prea mare, o rețea neurală va folosi mai mulți, ieșirile unor neuroni constituind intrări pentru alți neuroni. Neuroni vor comunica între ei prin activarea/lipsa activării, astfel imitând transmiterea de semnale electrice din cadrul creierului uman.

O rețea neurală cu mai multe straturi de neuroni mai este numită și *MultiLayer-Perceptron*, deși de obicei sunt folosiți neuroni de tip sigmoid, nu perceptron. O rețea de tip este organizată tipic pe mai multe straturi, având următoarea structură:

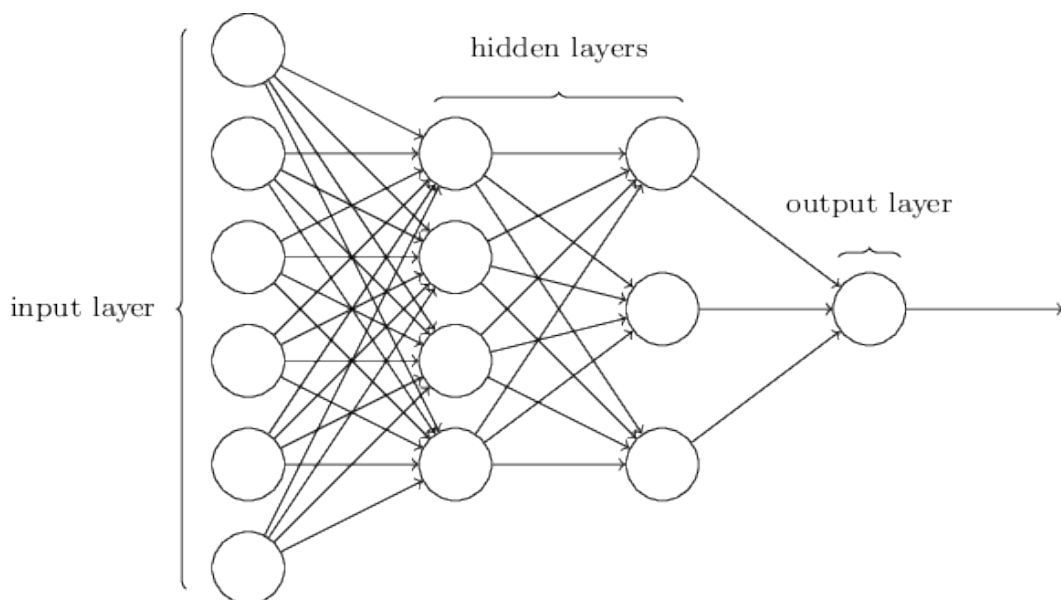


Figura 1.2: Structura unei rețele[9]

Straturile rețelei:

1. Stratul de intrare (*input*)

Stratul de *input* este format din neuroni de intrare, care preiau datele de intrare și le transmit mai departe rețelei. De cele mai multe ori, neuronii de intrare nu au ponderi și nici funcții de activare, având un rol pasiv în cadrul rețelei.

2. Stratul/Straturile de procesare (*hidden*)

Straturile *hidden* se ocupă de procesarea datelor de intrare. Există diferite moduri de a organiza straturile ascunse ale rețelei, în funcție de tipul rețelei: în cadrul unei rețele *feedforward* doar vor transmite datele prelucrate, iar în cadrul unei rețele convoluționale ce se ocupă de procesarea imaginilor vor exista straturi convoluționale și de *pooling* pentru a descoperi anumite trăsături în imagine, iar la rețelele de tip recurent, în straturile ascunse vor apărea conceptele de memorie și stări.

În toate cazurile, ideea de bază este ca straturile *hidden* vor prelua datele de intrare neprelucrate de la stratul de input, apoi vor prelucra acele date conform ponderilor asiguate și calculate în diverse moduri, pasând apoi datele procesate ultimului strat, cel de ieșire. Numărul de straturi *hidden*, organizarea lor, precum și tipurile de straturi și neuroni folosite depind atât de structura rețelei cât și de scopul în care este folosită aceasta.

3. Stratul de ieșire (*output*)

Neuronii din stratul de ieșire pot fi proiectați diferit față de straturile anterioare, reprezentând actorii finali ai rețelei și având rolul de a produce outputul programului. Astfel, stratul de ieșire se coalizează și produce în mod concret rezultatul final, cu rol în eficientizarea și îmbunătățirea acestuia. Referitor la cele menționate anterior, pentru a înțelege mai bine rețeaua neurală, cele trei nivele de straturi(stratul de intrare, straturile ascunse și stratul de ieșire)vor fi privite împreună ca întreg.

Pe lângă straturile descrise, o rețea neuronală mai are o funcție de cost folosită pentru a aproxima diferența dintre valoarea prezisă de rețea și valoarea reală. Mecanismul de antrenare utilizat la perceptron nu funcționează și aici, deoarece eroarea nu se propagă decât pe ultimul strat ascuns. Antrenarea unei rețele neurale are ca scop principal reducerea valorii funcției de cost, existând mai multe tehnici pentru a îmbunătăți acest procedeu. Un rol foarte important în antrenarea unei rețele îl are algoritmul de *Backpropagation*.

Backpropagation Când antrenăm un simplu perceptron, îi putem modifica instant ponderile în funcție de eroarea obținută (valoarea funcției de cost). În cazul unei rețele eroarea trebuie corectată la toate nivelele, nu doar la ultimul. Aici intervine algoritmul de *Backpropagation*. Algoritmul funcționează astfel:[4]

Pentru fiecare strat l , începând cu ultimul:

Pentru fiecare neuron i de pe stratul l :

1. Calculăm eroarea neuronului i de pe stratul l
2. Folosind eroarea calculată calculăm derivatele parțiale ale funcției de cost în funcție de pragul neuronului i și ponderile sale (derivatele parțiale ne arată cum variază funcția de cost în funcție de parametrii aleși)
3. Modificăm ponderile și/sau pragul pentru a corecta eroarea pentru acel neuron
4. Propagăm eroarea către neuronii de pe stratul $l-1$ și repetăm pașii de mai sus

Atât în timpul antrenării, cât și în timpul folosirii unei rețele (trecerea datelor de intrare prin rețea pentru a obține date de ieșire), se folosește foarte mult înmulțirea de matrici (pentru a înmulți ponderile neuronilor cu datele de intrare). Pentru anumite cazuri de input, putem optimiza numărul de parametri și procesul de funcționare. În continuare vom prezenta un tip de rețea optimizată în mod special pentru imagini.

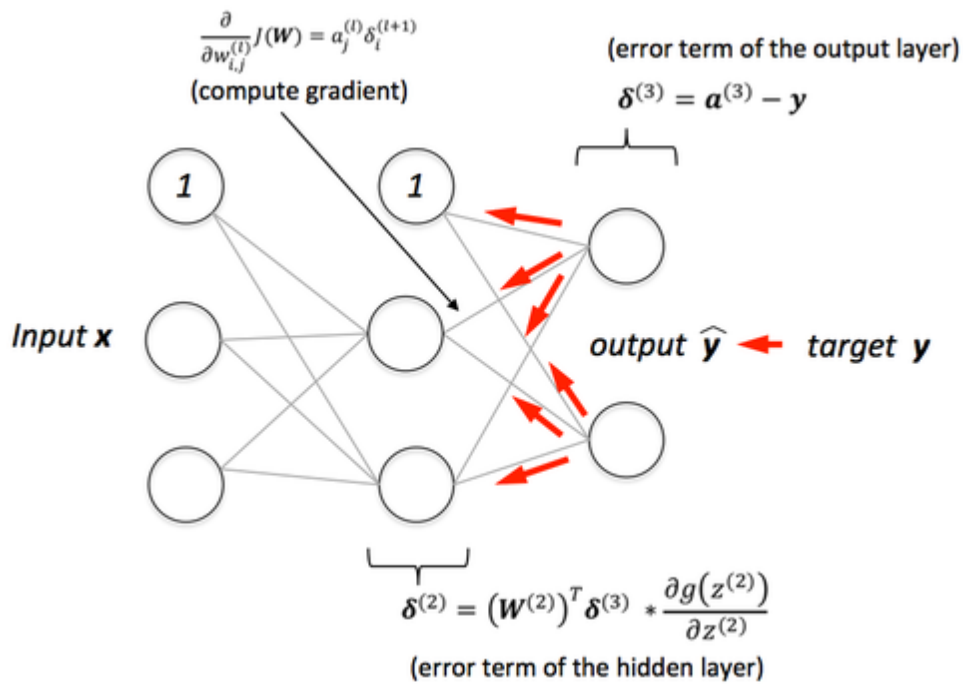


Figura 1.3: Algoritm de Backpropagation[10]

1.2 Rețele neurale convoluționale

O rețea neurală convoluțională este un tip specializat de rețea neurală pentru procesarea datelor organizate sub o formă de tablou (un tablou unidimensional pentru datele temporale, un tablou bidimensional de pixeli pentru imagini) care folosește o operație matematică specializată numită convoluție în locul înmulțirii generale de matrici.[6]

O rețea convoluțională are anumite tipuri de straturi care nu apar într-o rețea clasică: straturi convoluționale și straturi de *pooling*, dar la final apar și straturi conectate (*fully-connected*) similare cu cele clasice. În cadrul straturilor convoluționale și celor de *pooling* neuronii din fiecare strat vor fi conectați doar la o mică regiune din stratul precedent, în loc de conexiuni între toți neuronii cum aveam la o arhitectură clasică.

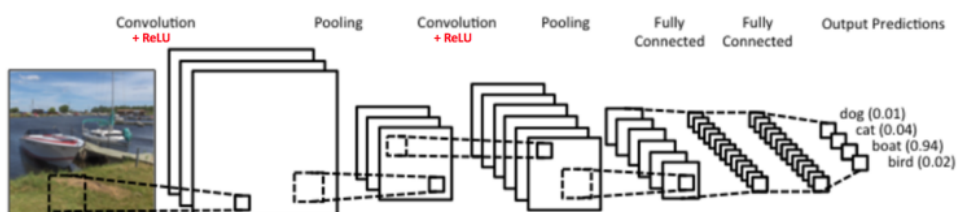


Figura 1.4: Arhitectura unei rețele convoluționale[1]

1.2.1 Convoluția

În cadrul unui strat de convoluție se folosește un filtru(sau *kernel*) pentru a extrage trăsături din cadrul imaginii. Filtrul se aplică unei mici regiuni din stratul precedent, realizând o mapare a imaginii în funcție de acel filtru. Cum imaginea este un tablou bidimensional de pixeli, un filtru va fi reprezentat tot de către un tablou bidimensional, mai mic decât cel de intrare. Aplicarea filtrului presupune calcularea produsului scalar între o parte din imagine de mărimea filtrului și filtru, efectuând această operație pentru toate părțile de aceeași mărime din imagine, luate la rând.

Un exemplu de aplicare a unui filtru pe o imagine:

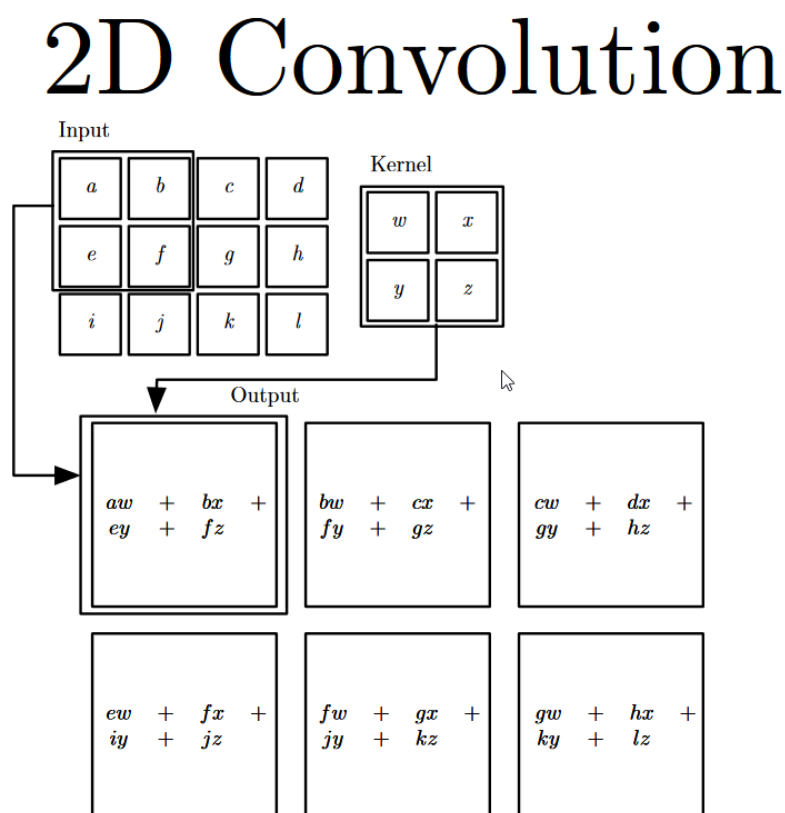


Figura 1.5: Operația de convoluție [6]

1.2.2 Pooling

Straturile de *pooling* (denumite și straturi de *subsampling* sau *downsampling*) vor prelua câte o mapare a imaginii dată de un filtru convoluțional și vor reduce dimensiunea fiecărei mapări, extrăgând astfel trăsăturile dominante. Pot exista mai multe tipuri de straturi de *pooling*: *MaxPooling*, *AveragePooling*, etc. De exemplu, în cazul operației

de *MaxPooling* ne alegem o zonă din mapare, din care extragem doar elementul maxim, astfel reducând dimensiunea mapării.

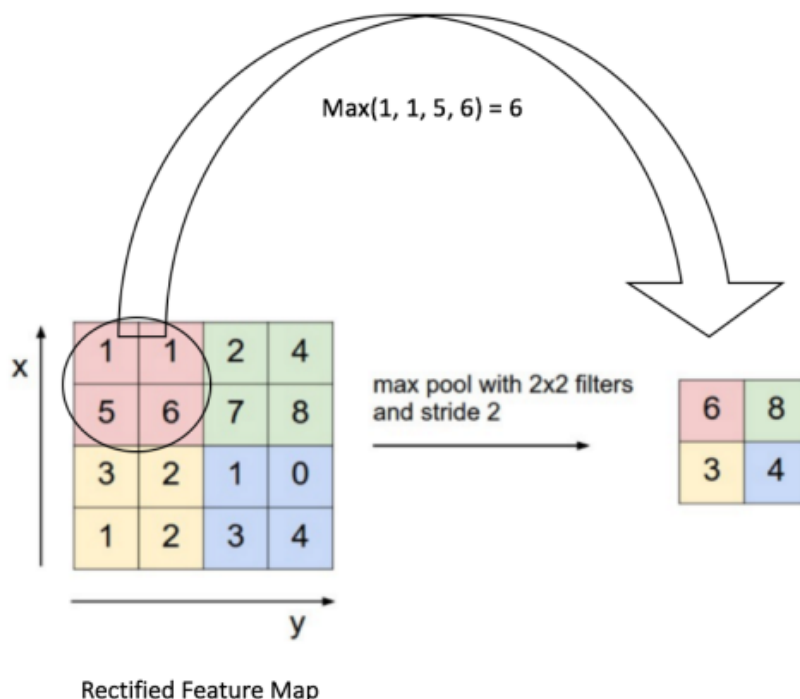


Figura 1.6: Exemplu de *MaxPooling*[1]

1.2.3 Clasificare

Adăugarea unui strat *fully-connected* ajută la clasificarea imaginii. Stratul de clasificare este de fapt un *MLP* tradițional, cu toți neuronii din el având conexiuni cu toți neuronii din stratul precedent. Stratul de clasificare va prelua trăsăturile de nivel înalt din imagine furnizate de straturile convoluționale și reduce ca dimensiune de straturile de pooling și va realiza o clasificare a imaginii bazată pe trăsăturile întâlnite.

Există multe tipuri de arhitecturi de rețele convoluționale, însă mai toate își au bazele pe operațiile descrise: convoluție, scalarea mapării (*pooling*), apoi clasificare.

1.3 Keras și Tensorflow

1.3.1 Tensorflow

Tensorflow este un *framework* cu sursă deschisă dezvoltat de Google și folosit pentru calcule numerice. Tensorflow oferă suport pentru învățare automată și *deep*

learning. Folosind diverse instrumente puse la dispoziție, utilizatorii pot construi modele de învățare automată la nivele diferite de abstractizare, de la modele construite pe baza unor serii de operații matematice la API-uri de nivel înalt pentru construirea de arhitecturi întregi disponibile, cum ar fi rețele neurale sau regresori liniari.

1.3.2 Keras

Keras reprezintă un API de nivel foarte înalt dedicat rețelelor neurale, scris în Python și care poate rula pe mai multe platforme de învățare automată cum ar fi TensorFlow, CNTK sau Theano. A fost dezvoltat pentru a abstractiza la maximum procesul de creare a unei rețele neurale, având ca scop atât experimentarea rapidă, cât și construcția într-un mod facil a unei arhitecturi solide. Keras poate fi văzut ca o interfață simplă, consistentă, folosită pentru cazurile uzuale, ce se pliază peste un *back-end* optimizat de *deep learning* și învățare automată. Este extrem de modular, permițând reutilizarea de blocuri configurabile, dar și extensibil, utilizatorii având posibilitatea de a crea noi straturi, funcții de cost sau arhitecturi pentru idei de cercetare.

1.4 OAuth

OAuth este un protocol deschis ce permite o metodă pentru o aplicație client să acceseze diverse resurse protejate de pe un server fără ca utilizatorul să își dezvăluie credențialele de pe server aplicației client.[8]

Procesul de autentificare OAuth folosește două tipuri de *token*[3]:

1. *Request token*

Folosit de aplicația client pentru a cere utilizatorului să autorizeze accesul la resursele protejate. *Token*-ul autorizat de utilizator este folosit o singură dată, este recomandat să expire după o limită de timp și va fi schimbat cu un *token* de acces.

2. *Access token*

Folosit de aplicația client să acceseze resursele protejate în numele utilizatorului.

Pașii folosiți în protocolul OAuth:

1. Aplicația trimite o cerere semnată pentru a primi un *Request token*
2. Serviciul furnizează aplicației *token*-ul solicitat

3. Aplicația redirectează utilizatorul spre o pagină de autorizare, unde utilizatorul își dă acordul, apoi se revine în contextul aplicației
4. Printr-o nouă cerere semnată, aplicația schimbă *Request Token*-ul cu un *token* de acces și un secret al acestui *token*
5. Aplicația folosește *token*-ul de acces și secretul acestuia pentru a accesa resursele protejate

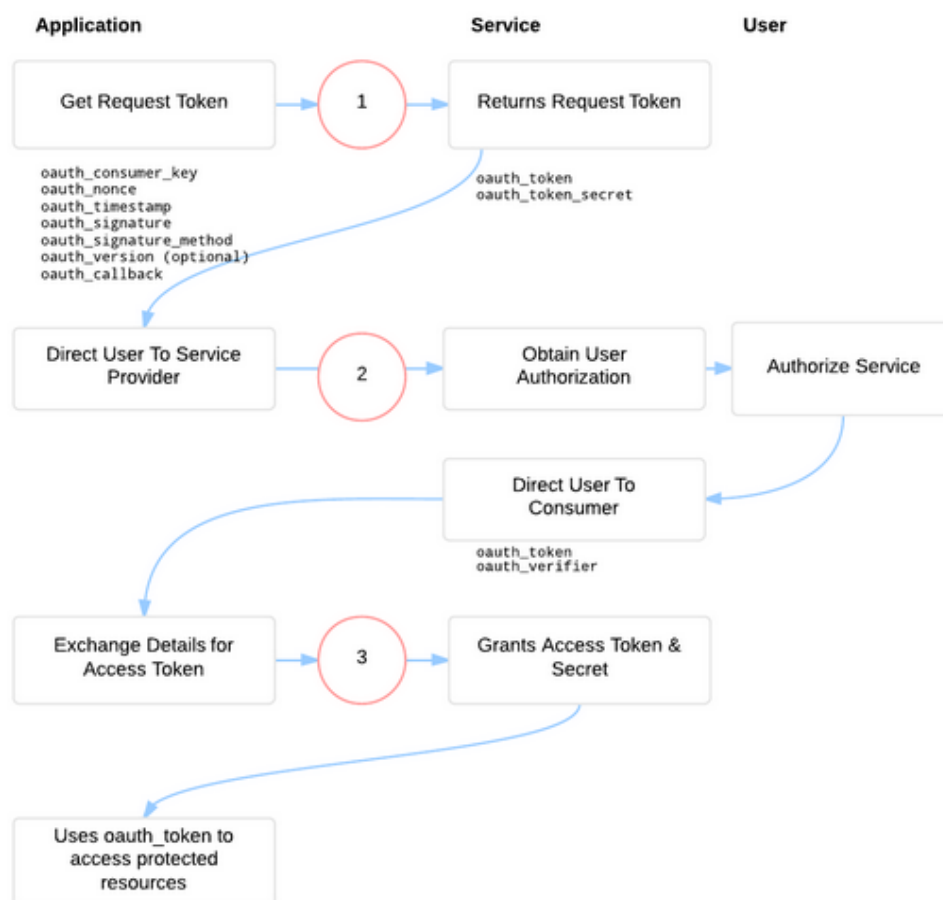


Figura 1.7: Autentificarea prin protocolul OAuth[2]

1.5 Angular

Angular este un *framework* cu sursă deschisă dezvoltat de către Google pentru crearea de aplicații client SPA (*single-page application*) folosind HTML, CSS și TypeScript. O aplicație Angular este compusă din module, componente și servicii.

1.6 Flask

Flask este un *microframework* WSGI(*Web Server Gateway Interface*) pentru realizarea de aplicații web, scris în Python. Este cunoscut pentru simplitatea sa, dar și pentru posibilitatea de a se scala pentru aplicații oricât de complexe.

Capitolul 2

Scopuri și cerințe

2.1 Necesitatea aplicației

Aplicația "FlickrCloneFetch" permite utilizatorului să vizualizeze imagini similare vizual cu o imagine încărcată, având posibilitatea selecției sursei imaginilor: propriul profil de Flickr, conturile contactelor sale, sau imagini relevante publice de pe Flickr. Astfel, utilizatorul obține o selecție de imagini similare prin prisma conținutului, ci nu al altor meta-date(locația imaginii, data la care a fost realizată, etc).

Un exemplu în care "FlickrCloneFetch" se dovedește utilă este următorul: Utilizatorul dorește să revadă imaginile cu castelele vizitate postate pe profilul său de Flickr. Deoarece vizitele au fost la intervale diferite de timp, este clar că o căutare după o perioadă de timp nu îi va răspunde cerințelor. De asemenea, cum castelele se află în locuri diferite nici o căutare după locație nu va corespunde obiectivului. O căutare după titlu va eșua din nou, deoarece utilizatorul și-a numit fotografiile folosind doar numele castelelor vizitate, evident diferite. Folosind aplicația "FlickrCloneFetch", utilizatorul va încărca o imagine cu un castel și va selecta faptul că dorește să vadă doar imagini de pe propriul profil. Aplicația va analiza imaginile de pe profilul său, afișând doar cele în care apar castele și astfel răspunzând cerinței utilizatorului.

2.2 Modul în care aplicația rezolvă problema vizată

Utilizatorul se va autentifica în aplicație cu contul său de Flickr, astfel oferind aplicației acces la imaginile personale și la lista de contacte. După autentificare, utilizatorul va încărca o imagine în aplicație și va selecta sursa imaginilor ce vor fi analizate

(în mod general, sursa este profilul personal de Flickr).

Imaginea încărcată este analizată cu ajutorul unei rețele neurale, analiză în urma căreia se obțin anumite etichete ce descriu conținutul imaginii. Imaginile din sursa selectată de utilizator vor fi la rândul lor analizate cu aceeași rețea, etichetele obținute fiind comparate cu lista inițială de etichete, similaritatea conținutului fiind detectată prin existența etichetelor identice în imagini diferite.

Imaginile care au etichete ce apar și în lista de etichete a imaginii încărcate vor fi afișate în aplicație.

2.3 Comparația cu soluții existente

O soluție ce la prima vedere pare că ar rezolva problema expusă ar fi chiar căutarea oferită de *Flickr*. Însă, căutarea după anumite cuvinte cheie se realizează verificând existența cuvintelor în titlul, descrierea și lista de etichete ale imaginii. Astfel soluția de căutare nu oferă garanția faptului că ceea ce căutăm se află chiar în conținutul imaginii, titlul, descrierea și etichetele fiind alese de către utilizator după preferințele sale.

2.3.1 SimilarityPivot

SimilarityPivot este o unealtă oferită de Flickr care folosește rețele neurale pentru a analiza imaginile și a găsi imagini care au o anumită mărime, orientare sau paletă de culori. SimilarityPivot se poate combina cu căutarea după cuvinte cheie pentru a returna imagini ce conțin în titlu, descriere sau etichete acele cuvinte și în același timp respectă și cerințele vizuale descrise mai sus. Diferența între SimilarityPivot și "FlickrCloneFetch" constă în faptul că SimilarityPivot analizează în mod general imaginea, căutând doar anumite criterii vizuale (precum paleta de culori), în timp ce "FlickrCloneFetch" analizează imaginea pentru a descoperi etichete care să îi reprezinte conținutul. Astfel, cele două aplicații prezintă scenarii de utilizare diferite.

2.3.2 Reverse image search

Există mai multe sit-uri ce oferă posibilitatea de a efectua o căutare inversă după imagini, cel mai cunoscut pentru această funcționalitate fiind chiar Google. Funcționalitatea oferită de Google oferă utilizatorului posibilitatea să încarce o imagine, analizează imaginea și returnează o etichetă ce descrie conținutul imaginii. De asemenea, prezintă sit-

uri care corespund etichetei returnate și afișează imagini similare din punct de vedere vizual. Totuși, spre deosebire de o căutare uzuală, pentru acest tip de căutare nu se poate restrânge domeniul căutării la un singur sit, deci imaginile similare vor fi aduse de pe întreg web-ul, nu de pe o platformă specificată. De asemenea, căutarea poate găsi doar imagini publice, nu și imagini vizibile doar unui anumit utilizator (de pe profilul său, sau imagini ale contactelor cu vizibilitate doar pentru lista de prieteni). Astfel, diferența între soluțiile de căutare inversă după imagini și "FlickrCloneFetch" constă în domeniul căutării, precum și posibilitatea "FlickrCloneFetch" de a accesa fotografii vizibile doar unui utilizator autentificat.

2.3.3 Servicii pentru analiza imaginilor

Platformele de *cloud computing* oferă și servicii de analiză vizuală a imaginilor, precum CloudVision de la Google Cloud Platform, ComputerVision de la AmazonWeb-Services, și ComputerVision de la Microsoft Azure Cloud. Toate aceste servicii pot fi folosite pentru a realiza aplicații similare "FlickrCloneFetch", însă nu reprezintă în sine o soluție la problema expusă.

Capitolul 3

Analiză și proiectare

3.1 Scenarii de utilizare

Utilizatorul poate efectua următoarele acțiuni:

3.1.1 Logare

Utilizatorul se loghează în aplicație, care îl va redirecționa către platforma Flickr, pentru a efectua autentificarea prin intermediul acesteia.

3.1.2 Alege sursa imaginilor

Utilizatorul poate alege sursa imaginilor ce vor fi analizate: propriul profil, imaginile contactelor sale, sau imagini publice de pe platforma Flickr.

3.1.3 Încarcă imagine

Pentru a efectua această acțiune, utilizatorul trebuie să fi trecut prin pasul de logare și eventual să fi ales sursa imaginilor. Dacă acesta nu a ales sursa imaginilor, în mod implicit aceasta este propriul profil. Scenariul acesta de utilizare presupune ca utilizatorul să aleagă o imagine de pe propriul dispozitiv pe care să o trimită către aplicație.

Pentru a returna imaginile similare din punct de vedere vizual, aplicația va efectua următoarele acțiuni:

1. Analizează imaginea încărcată folosind o rețea neurală și returnează o serie de etichete descriptive

2. Aduce imaginile din sursa selectată de utilizator
3. Analizează imaginile aduse, obținând pentru fiecare o serie de etichete
4. Descoperă similaritatea între imagini comparând etichetele și obține o listă de imagini corespunzătoare
5. Afișează imaginile din lista obținută

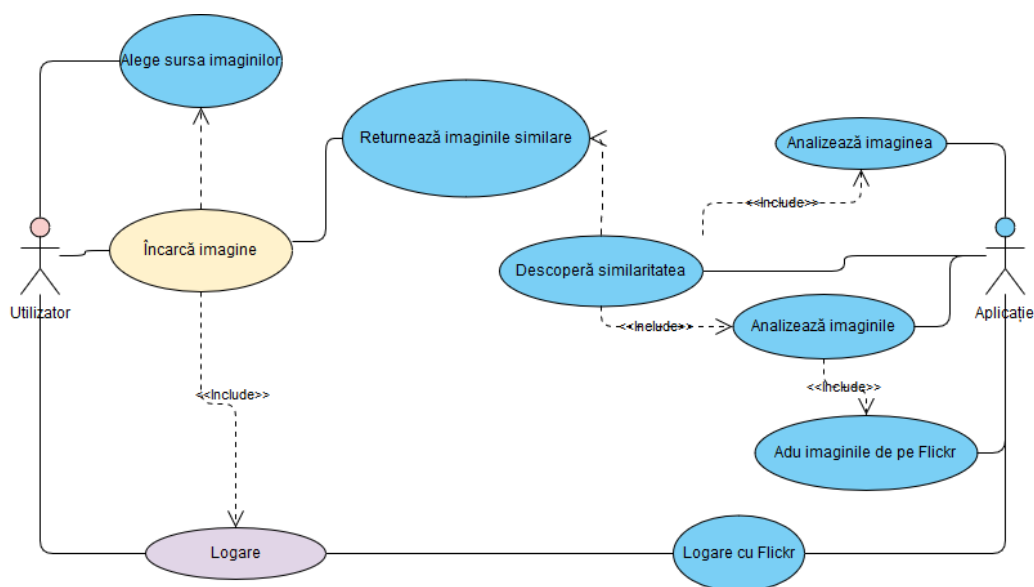


Figura 3.1: Diagrama de scenarii de utilizare a aplicației

3.2 Arhitectura aplicației

Aplicația este împărțită în două module, dar folosește și anumite servicii externe. Utilizatorul interacționează direct cu modulul *front-end*, o aplicație web realizată cu ajutorul *framework*-ului Angular în limbajul Typescript. La rândul său, modulul este împărțit în mai multe componente, acestea apelând unele servicii. Modulul de back-end este realizat cu ajutorul *framework*-ului Flask în limbajul Python. Se folosesc mai multe biblioteci pentru a efectua anumite operații (ex. *requests* pentru a apela API-uri, *pony.orm* pentru conectarea la baza de date) și *framework*-uri specializate pentru rețele neuronale (Tensorflow și Keras). Serviciile externe sunt reprezentate de API-urile platformelor Flickr (pentru autentificare, informații despre profil și imagini) și Data-Muse (pentru a obține sinonime). De asemenea se mai folosește o bază de date SQLite pentru reținerea *token*-ului de acces și identificatorului unic.

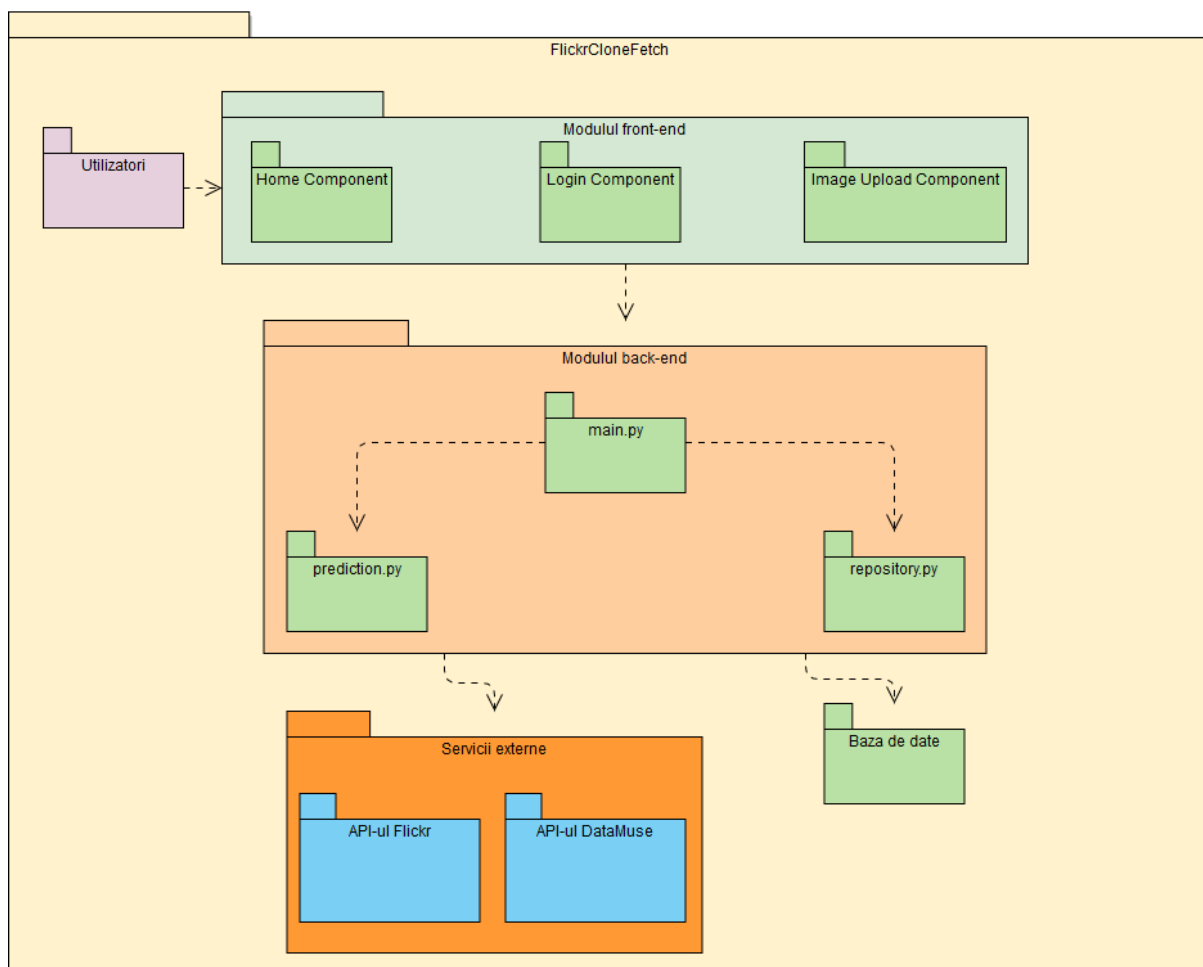


Figura 3.2: Arhitectura aplicației

3.3 Planificarea activităților realizate de către aplicație

Fluxul de activități în cadrul aplicației se desfășoară în modul urmator

1. Modulul *front-end* afișează pagina de logare și redirecționează utilizatorul la procesul de logare de pe *back-end* (Home Component)
2. Modulul *back-end* trimite cererea de logare prin OAuth la API-ul Flickr
3. Se salvează *request token*-ului și redirecționează la autorizarea pagina de logare în Flickr și autorizare prin API-ul Flickr
4. Se trimite *request token*-ul autorizat de către utilizator pentru a obține un *token* de acces
5. Se generează un identificator unic pentru utilizator, se salvează în baza de date împreună cu *token*-ul de acces primit și apoi se face redirecționează către *front-end* trimițând și identificatorul unic
6. Modulul *front-end* reține identificatorul utilizatorului și cere datele despre profil
7. Modulul *back-end* furnizează datele despre profil (nume utilizator, imagine de profil)
8. *Front-end*-ul redirecționează utilizatorul la pagina de încărcare imagini (Login Component)
9. Aici i se oferă utilizatorului posibilitatea de a încărca o imagine și de a alege sursa imaginilor, apoi trimite la modulul *back-end* aceste informații (Image Upload Component)
10. Modulul *back-end* primește imaginea și opțiunile privitoare la sursă și analizează imaginea cu o rețea neurală, obținând o listă de etichete
11. Se apelează API-ul DataMuse pentru a obține cuvinte similare etichetelor obținute
12. Se apelează API-ul Flickr pentru a obține imagini și acestea se analizează cu rețeaua neurală
13. Se compară etichetele imaginilor cu etichetele imaginii încărcate și se creează o listă de imagini similare

14. Se returnează lista către *front-end*

15. *Front-end*-ul afișează imaginile returnate de *back-end* (Image Upload Component)

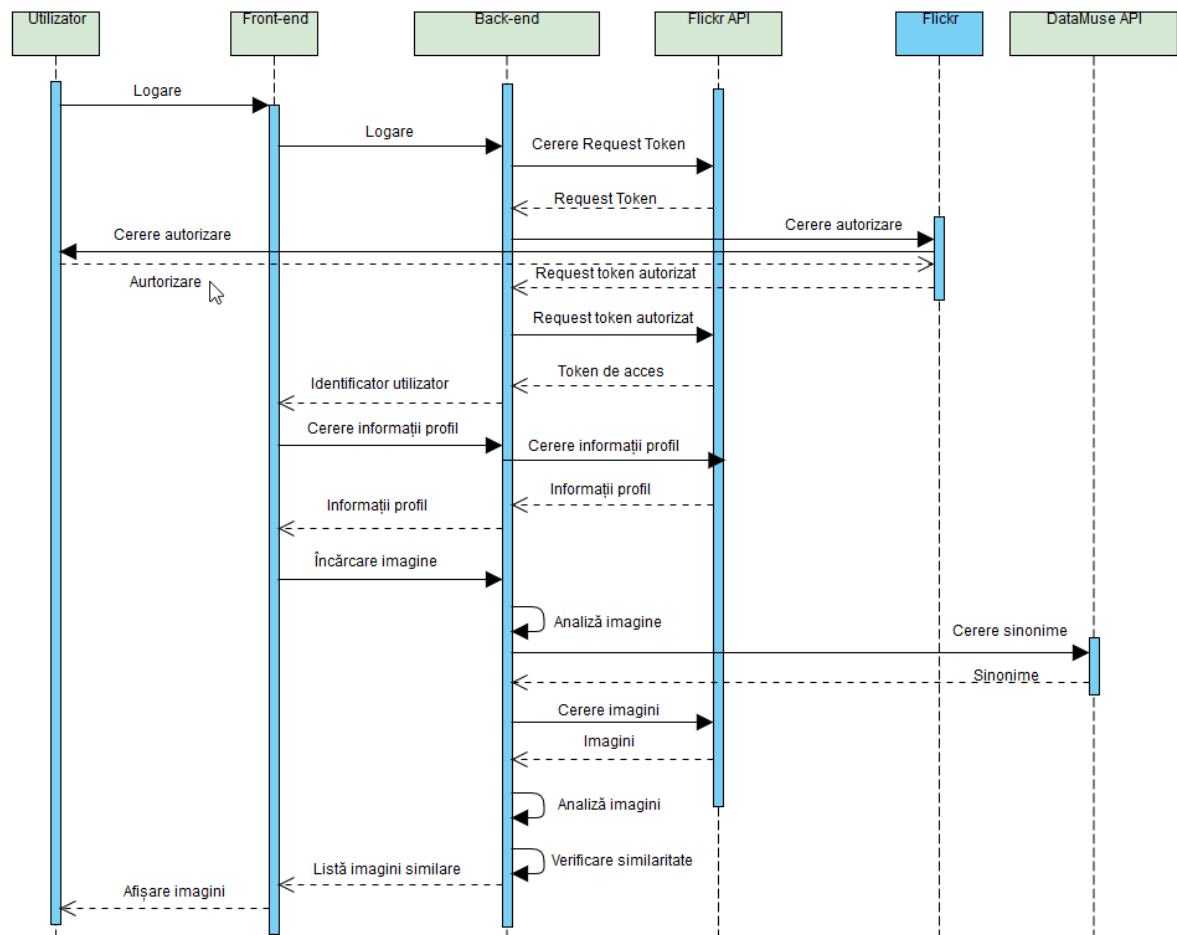


Figura 3.3: Diagrama de secvență a aplicației

3.4 Modulul *back-end*

3.4.1 API-ul expus

Specificația OPENApi a API-ului se găsește la Anexa 1

/login

Endpoint pentru logare, utilizatorul este redirectat aici de către modulul *front-end*. Aplicația va redirecta la rândul ei către /authorize.

/authorize

Endpoint pentru autorizare, se ajunge aici din redirectarea efectuată de către aplicație. Are ca parametrii *oauth-token* și *oauth-verifier* și va redirecționa către *front-end*.

/profile

Returnează informațiile despre profilul de Flickr al utilizatorului: numele utilizatorului și imaginea sa de profil. Are ca parametru identificatorul utilizatorului.

/addImage

Primește identificatorul utilizatorului, imaginea în base64 și opțiunea privitoare la sursa imaginilor ce vor fi analizate. Returnează lista de etichete a imaginii și lista de imagini similare din punct de vedere vizual.

3.4.2 Rețeaua neurală folosită

Rețeaua neurală folosită pentru analiza imaginilor este un model numit *NASNet Mobile*, obținut de Google prin *framework*-ul NAS (Neural Architecture Search) și discutat în lucrarea *Learning Transferable Architectures for Scalable Image Recognition*. [12]

Proiectul *AutoML*, prin *framework*-ul NAS, propune următorul lucru: o rețea neurală controler propune o arhitectură de rețea "copil" care va fi antrenată și evaluată pe o anumită sarcină. Evaluarea este folosită pentru a calibra rețeaua controler, care va propune idei îmbunătățite de arhitectură la iterația următoare. Procesul se repetă de mii de ori, generând noi arhitecturi, testându-le și evaluându-le până când se obțin arhitecturi de rețele neurale noi utile.[11]

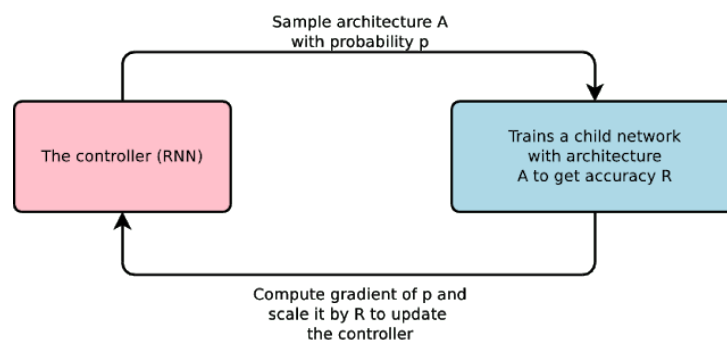


Figura 3.4: Procesul de obținere a noi rețele neurale folosind NAS [11]

Arhitectura *NASNet* este obținută prin procesul descris mai sus, cu unele mici modificări. Scopul rețelei e să clasifice imagini, deci a fost evaluată performanța pe două din cele mai cunoscute seturi de date în mediul academic: setul de detecție de obiecte *COCO* și setul de clasificare de imagini *ImageNet*. Deoarece seturile acestea de date sunt foarte mari, AutoML a fost reprogramat să caute straturi care ar fi atunci când sunt conectate în număr mare. Căutarea straturilor a avut loc pe un set de date mai mic, CIFAR10, apoi cea mai bună arhitectură a fost scalată și transferată pentru antrenamentul pe *COCO* și *ImageNet*. [12]

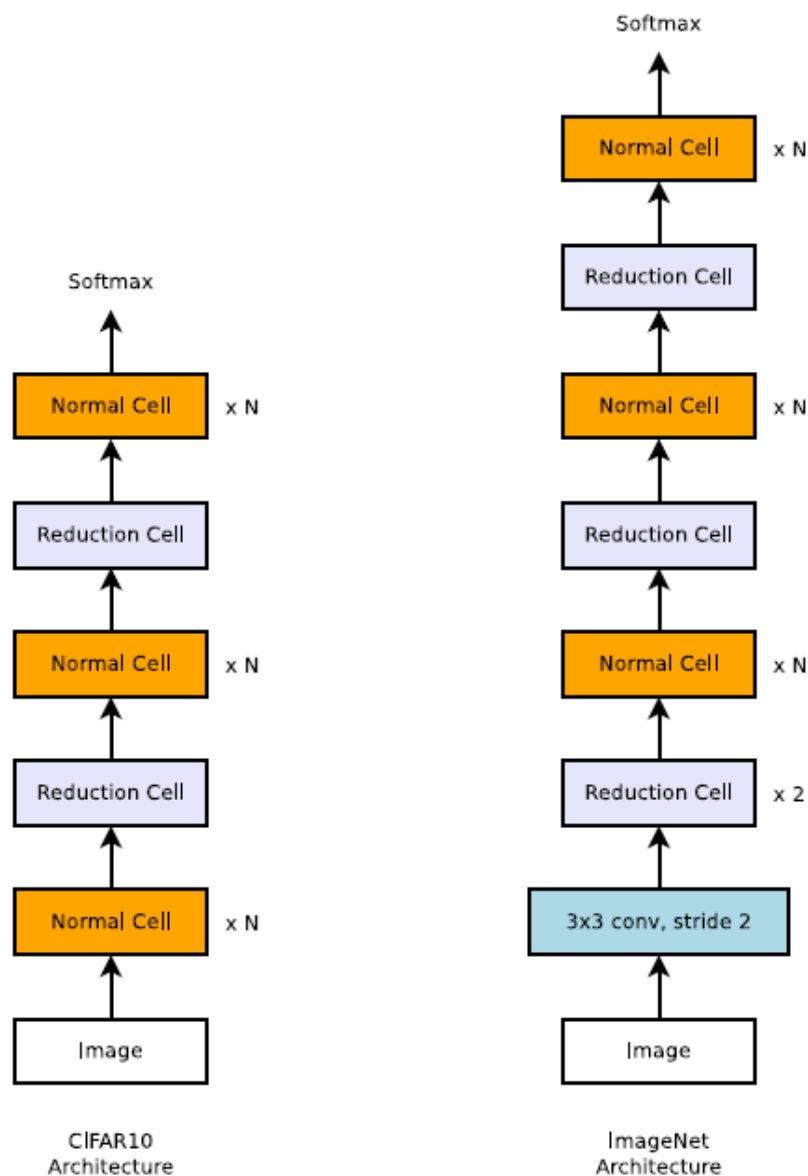


Figura 3.5: Arhitectura NASNet [12]

Arhitectura *NASNet* a fost predefinită, având două tipuri de straturi:

1. *Normal Cell* - un strat convoluțional care returnează o mapare de aceeași dimensiune a imaginii
2. *Reduction Cell* - un strat convoluțional care reduce dimensiunea mapării cu un factor de doi

Rețeaua neurală controler a trebuit doar să descopere cea mai bună arhitectură pentru cele două tipuri de straturi.

După aproximativ 4 zile de căutări au fost obținute mai multe arhitecturi de straturi candidat, care combinate produc arhitecturi cu rezultate comparabile cu alte arhitecturi din domeniu. Cele mai bune 3 arhitecturi obținute se numesc *NASNet-A*, *NASNet-B* și *NASNet-C*.

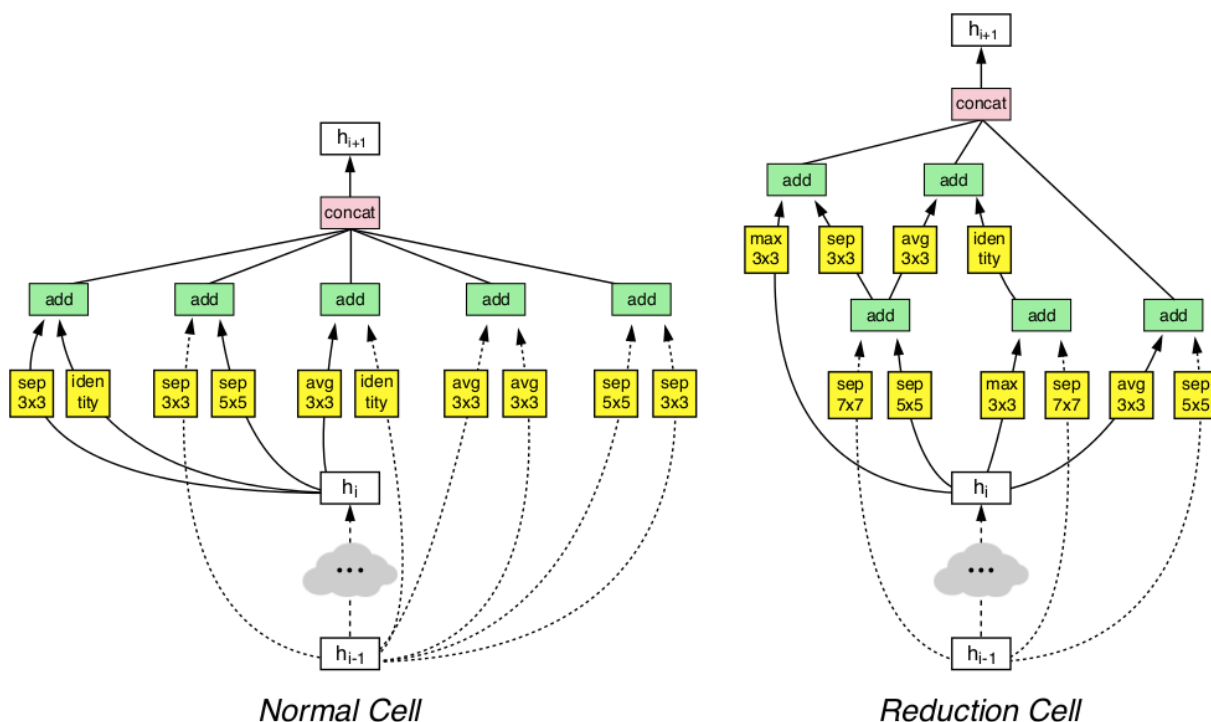



Figura 3.6: Celulele arhitecturii NASNet-A [12]

Modelul folosit în aplicația "FlickrCloneFetch", numit *NASNet Mobile*, este o versiune scalată a arhitecturii *NASNet*, mai mică, cu 5 326 716 parametri optimizată pentru un cost computațional redus și o viteză mai mare. Modelul folosit este pre-antrenat pe 1000 de categorii din setul de date ImageNet, având o acuratețe de 91.9 procente pe același set de date.

3.4.3 Baza de date

Baza de date folosită, de tip SQLite, are un singur tabel folosit pentru a stoca *token*-ul de acces și identificatorul unicat generat în relație cu acesta.



| users | |
|--------------------|---------|
| user_nsid | int |
| user_id | int |
| username | varchar |
| fullname | varchar |
| oauth_token | varchar |
| oauth_token_secret | varchar |

Figura 3.7: Schema bazei de date folosite

Capitolul 4

Detalii de implementare

Concluzii

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nunc mattis enim ut tellus elementum sagittis vitae et. Placerat in egestas erat imperdiet sed euismod. Urna id volutpat lacus laoreet non curabitur gravida. Blandit turpis cursus in hac habitasse platea. Eget nunc lobortis mattis aliquam faucibus. Est pellentesque elit ullamcorper dignissim cras tincidunt lobortis feugiat. Viverra maecenas accumsan lacus vel facilisis volutpat est. Non odio euismod lacinia at quis risus sed vulputate odio. Consequat ac felis donec et odio pellentesque diam volutpat commodo. Etiam sit amet nisl purus in. Tortor condimentum lacinia quis vel eros donec. Phasellus egestas tellus rutrum tellus pellentesque eu tincidunt. Aliquam id diam maecenas ultricies mi eget mauris pharetra. Enim eu turpis egestas pretium.

Bibliografie

- [1] ***. An intuitive explanation of convolutional neural networks. <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>.
- [2] ***. Oauth bible. <http://oauthbible.com/>.
- [3] ***. Oauth core 1.0 revision a. <https://oauth.net/core/1.0a/>.
- [4] Răzvan Benchea. Neural networks. <https://sites.google.com/view/rbenchea/neural-networks?authuser=0>.
- [5] Adi Chris. From perceptron to deep neural nets. <https://becominghuman.ai/from-perceptron-to-deep-neural-nets-504b8ff616e>.
- [6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [7] Kevin Gurney. *An Introduction to Neural Networks*. Taylor & Francis, Inc., 1997.
- [8] Eran Hammer-Lahav. The oauth 1.0 protocol. RFC 5849, RFC Editor, April 2010. <http://www.rfc-editor.org/rfc/rfc5849.txt>.
- [9] Michael A. Nielsen. *Neural Networks and Deep Learning*. De3termination Press, 2015.
- [10] Sebastian Raschka. A visual explanation of the back propagation algorithm for neural networks. <https://www.kdnuggets.com/2016/06/visual-explanation-backpropagation-algorithm-neural-networks.html>.
- [11] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. 2017.

- [12] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017.

Anexa 1

```
{
  "openapi": "3.0.2",
  "info": {
    "title": "Back-end",
    "version": "1.0.0"
  },
  "paths": {
    "/login": {
      "summary": "Endpoint logare",
      "description": "Enpoint-ul pentru logare, utilizatorul  

        ↪ este redirectat aici din browser.",
      "get": {
        "responses": {
          "302": {
            "description": "Redirectioneaza catre /authorize  

              ↪ , incluzand oauth_token si  

              ↪ oauth_verifier."
          }
        },
        "summary": "GET pentru logare",
        "description": "GET din browser, utilizatorul fiind  

          ↪ redirectat aici din front-end."
      }
    },
    "/authorize": {
      "summary": "Endpoint pentru autorizare",
```

```

"description": "Endpoint la care se ajunge din
    ↪ redirectarea efectuata de aplicatie.",
"get": {
    "responses": {
        "302": {
            "description": "Redirectioneaza catre front-end
                ↪ ."
        }
    },
    "summary": "GET autorizare"
},
"parameters": [
    {
        "name": "oauth_token",
        "description": "Tokenul OAuth",
        "schema": {
            "type": "string"
        },
        "in": "query",
        "required": true
    },
    {
        "name": "oauth_verifier",
        "description": "",
        "schema": {
            "type": "string"
        },
        "in": "query",
        "required": true
    }
]
},
"/addImage": {

```

```

"summary": "Adaugare imagine",
"description": "",
"post": {
  "requestBody": {
    "description": "Identificatorul utilizatorului,
    ↪ imaginea in base64 si sursa.",
    "content": {
      "application/json": {
        "examples": {
          "post-image": {
            "value": {
              "user": "id-ul utilizatorului",
              "file": "imaginea in base64",
              "option": "sursa imaginilor"
            }
          }
        }
      }
    },
    "required": true
  },
  "responses": {
    "200": {
      "content": {
        "application/json": {
          "examples": {
            "full-example": {
              "value": {
                "labels": [
                  "label1",
                  "label2"
                ],
                "images": [

```

```

        "url1",
        "url2"
    ]
}
},
"empty-example": {
    "value": {
        "labels": [
        ],
        "images": [
        ]
    }
}
}
},
},
"description": "Returneaza descrierea si
    ↪ imaginile."
},
"401": {
    "description": "Daca utilizatorul nu este
    ↪ logat/ nu exista."
},
"400": {
    "description": "JSON malformat/incomplet."
}
},
"summary": "POST imagine"
}
},
"/profile": {
    "description": "Returneaza informatiile despre
    ↪ profilul de Flickr al utilizatorului.",

```

```

"get": {
  "responses": {
    "200": {
      "content": {
        "application/json": {
          "examples": {
            "profil": {
              "value": {
                "name": "nume-user",
                "buddyicon": "url imagine de
                  ↳ profil"
              }
            }
          }
        }
      },
      "description": "Cazul in care utilizatorul
        ↳ exista si este logat."
    },
    "401": {
      "description": "Cazul in care utilizatorul nu
        ↳ este logat/ nu exista."
    }
  },
  "description": "Returneaza informatiile."
},
"parameters": [
  {
    "name": "user",
    "description": "Identificatorul utilizatorului",
    "schema": {
      "type": "string"
    }
  },

```



```
        "in": "query",
        "required": true
    }
]
}
}
```