



**ANALIZA AUTOMATĂ A EVOLUTIEI GHETARILOR
DIN IMAGINI SATELITARE FOLOSIND REȚELE
NEURONALE CONVOLUȚIONALE**

LUCRARE DE LICENȚĂ

Absolvent: **Rares MICLEA**

Coordonator **Prof. dr. ing. Florin-Ioan ONIGA**
științific:

2025



FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Vlad MUREŞAN

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Rares MICLEA**

**ANALIZA AUTOMATĂ A EVOLUȚIEI GHETARILOR DIN IMAGINI
SATELITARE FOLOSIND REȚELE NEURONALE CONVOLUTIONALE**

- Enunțul temei:** Lucrarea urmărește dezvoltarea unui sistem software pentru analiza automată a evoluției ghetarilor pe baza imaginilor satelitare. Solutia integrează un model de segmentare semantică, un backend pentru descărcarea și procesarea imaginilor, și o interfață web pentru vizualizarea rezultatelor. Aplicația permite identificarea și cuantificarea modificărilor glaciare în timp, fiind destinată monitorizării schimbărilor climatice.
- Conținutul lucrării:** Cuprins, Introducere, Obiectivele Proiectului, Studiu Bibliografic, Analiză și fundamentare teoretică, Proiectare de detaliu și implementare, Testare și validare, Manual de instalare și utilizare, Concluzii, Bibliografie.
- Locul documentării:** Universitatea Tehnică din Cluj-Napoca, Departamentul de Calculatoare
- Consultanți:**
- Data emiterii temei:** 1 Noiembrie 2024
- Data predării:** 11 Iulie 2025

Absolvent: _____

Coordonator științific: _____

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

**Declarație pe propria răspundere privind
autenticitatea lucrării de licență**

Subsemnatul Miclea Rareș, legitimat cu cartea de identitate seria SB nr. 959519, CNP 5030118324801, autorul lucrării ANALIZA EVOLUȚIEI GHETARILOR DIN IMAGINI SATELITARE FOLOSIND REȚELE NEURONALE CONVOLUȚIONALE, elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea Tehnologia Informației, din cadrul Universității Tehnice din Cluj-Napoca, sesiunea iulie a anului universitar 2024-2025, declar pe propria răspundere că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării și în bibliografie.

Declar că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

În cazul constatării ulterioare a unor declarații false, voi suporta sanctiunile administrative, respectiv, *anularea examenului de licență*.

Data

Miclea Rareș

11.07.2025



Semnătura

Cuprins

Capitolul 1 Introducere	1
Capitolul 2 Obiectivele proiectului	3
2.1 Obiectiv principal	3
2.2 Obiective secundare	3
2.2.1 Dezvoltarea unui model propriu de recunoaștere glaciară	3
2.2.2 Dezvoltarea aplicației software de analiză a ghețarilor	4
Capitolul 3 Studiu bibliografic	5
3.1 Introducere în segmentarea imaginilor satelitare	5
3.2 Arhitecturi CNN pentru segmentarea ghețarilor	5
3.2.1 U-Net și derivate	5
3.2.2 Modelul hibrid U-Net + Transformeri	6
3.3 Funcții de pierdere pentru segmentare semantică	6
3.3.1 Dice Loss și extinderi	6
3.3.2 Alte funcții de pierdere	6
3.4 Tehnici de preprocesare și augmentare	6
3.5 Evaluare și metriki	7
3.6 Rezumat comparativ și stadiul actual	7
3.7 Tehnologii moderne pentru dezvoltarea aplicațiilor geospațiale	7
3.7.1 Arhitecturi backend RESTful în context geospațial	7
3.7.2 Accesarea imaginilor satelitare: Sentinel Hub și protocoale OGC .	8
3.7.3 Framework-uri frontend pentru aplicații geospațiale interactive .	8
3.7.4 Comunicarea client-server și bune practici de integrare	9
3.7.5 Stadiul actual al tehnologiilor geospațiale în aplicații ML	10

Capitolul 4 Analiză și fundamentare teoretică	11
4.1 Structura generală a proiectului	11
4.2 Arhitectura modelului de detecție a ghețarilor	12
4.2.1 Encoder (Blocurile de extragere a caracteristicilor)	12
4.2.2 Bottleneck (Stratul de mijloc profund)	12
4.2.3 Decoder (Reconstrucția segmentării)	13
4.2.4 Strat final (Predictor segmentare)	13
4.3 Algoritmi și concepte utilizate în procesul de detecție a ghețarilor	13
4.3.1 Funcția de pierdere - Dice Loss	13
4.3.2 Optimizare și strategie de antrenare	14
4.3.3 Metrici de evaluare	14
4.4 Considerații privind variabilitatea datelor satelitare	15
4.4.1 Prezența norilor și a nebulozității subtile	15
4.4.2 Confuzia dintre gheată și zăpada recentă	15
4.4.3 Umbre topografice și iluminare neuniformă	15
4.4.4 Variații sezoniere și interanuale	16
4.5 Backend-ul aplicației	16
4.5.1 Coordonarea fluxului de analiză	16
4.5.2 Integrarea cu servicii externe de imagini satelitare	17
4.5.3 Apelarea modelului de segmentare și generarea rezultatelor	17
4.5.4 Persistența și expunerea rezultatelor	18
4.6 Frontend-ul aplicației	18
4.6.1 Principii fundamentale de proiectare	18
4.6.2 Fluxul informațional și legătura cu backend-ul	19
4.6.3 Vizualizarea datelor geospațiale și temporale	19
4.6.4 Arhitectura orientată pe componente	20

4.6.5	Principii de design vizual și experiență utilizator	20
4.6.6	Scalabilitate și mențenanță	20
Capitolul 5	Proiectare de detaliu și implementare	21
5.1	Implementarea modelului de detectie glaciara	21
5.1.1	Setul de date utilizat	21
5.1.2	Structura fișierelor și organizarea codului	21
5.1.3	Definirea blocurilor convolutionale reziduale	22
5.1.4	Functia <code>build_model</code> și integrarea arhitecturii	22
5.1.5	Strategii de initializare și reproducibilitate	22
5.1.6	Padding și alinierea ieșirilor convolutionale	23
5.1.7	Implementarea blocurilor encoder și decoder	23
5.1.8	Regularizare și controlul supraînvățării	23
5.1.9	Encoderul rețelei	24
5.1.10	Stratul bottleneck	24
5.1.11	Decoderul și reconstrucția segmentării	24
5.1.12	Debugging și testare incrementală	25
5.1.13	Gestionarea dezechilibrului între clase	25
5.1.14	Stratul final și predicția clasei	25
5.1.15	Sumar al arhitecturii modelului	26
5.1.16	Compilarea modelului și funcția de pierdere	27
5.1.17	Configurarea optimizatorului și strategia de învățare	27
5.1.18	Augmentarea și procesarea setului de antrenament	27
5.1.19	Antrenarea controlată și salvarea modelului	27
5.1.20	Evaluarea finală	27
5.1.21	Salvarea și exportul modelului	28
5.1.22	Generarea rezultatelor și salvarea imaginilor	28

5.2	Backend-ul aplicației	28
5.2.1	Structură generală și rol funcțional	28
5.2.2	Procesarea cererii de analiză	29
5.2.3	Selectia și descărcarea imaginilor Sentinel	29
5.2.4	Rularea modelului și generarea rezultatelor	29
5.2.5	Execuția asincronă și tratamentul erorilor	30
5.2.6	Gestionarea anilor indisponibili și fallback implicit	30
5.2.7	Considerații privind scalabilitatea și extensibilitatea	31
5.2.8	Securitate și auditabilitate	31
5.2.9	Fluxul complet al inferenței și salvarea metadatelor	31
5.2.10	Structura de directoare și persistarea rezultatelor	32
5.2.11	Formatul fișierului config.json	32
5.2.12	Endpoint-uri disponibile și interacțiune cu frontend-ul	33
5.2.13	Gestionarea statică a resurselor și performanță	33
5.3	Interfața frontend	34
5.3.1	Structura generală și tehnologii utilizate	34
5.3.2	Design vizual și integrare UI	35
5.3.3	Structura aplicației și organizarea componentelor	35
5.3.4	Modularitate și logică partajată	36
5.3.5	Descrierea componentelor	37
5.3.6	Exemplu detaliat: image-viewer.tsx	37
5.3.7	Exemplu detaliat: glacier-graph.tsx	38
5.3.8	Responsivitate și compatibilitate	38
5.3.9	Cazuri de utilizare ale interfeței frontend	38
5.3.10	Integrarea cu backend-ul și fluxul de date	40
5.3.11	Controlul stării și feedback-ul utilizatorului	40

5.3.12 Extensibilitate	41
Capitolul 6 Testare și validare	42
6.1 Scenarii de testare	42
6.2 Instrumente și metode de testare	42
6.3 Tipuri de teste efectuate	43
6.3.1 Teste unitare	43
6.3.2 Teste de integrare	43
6.3.3 Teste de acceptanță	43
6.3.4 Teste de uzabilitate	43
6.4 Testarea modulelor și fluxurilor complete	44
6.4.1 Testarea aplicației complete	44
6.5 Validarea rezultatelor modelului	44
6.5.1 Metrici utilizate și semnificația lor	44
6.5.2 Rezultate obținute	45
6.5.3 Comparație cu literatura de specialitate	47
6.6 Limitări ale testării efectuate	47
6.7 Robusteză, fiabilitate și reproductibilitate	48
6.7.1 Reexecutabilitatea completă a analizelor	48
6.7.2 Fiabilitatea în fața întreruperilor	48
6.7.3 Controlul versiunilor și stabilitatea codului	48
6.7.4 Testarea reproductibilității complete	49
Capitolul 7 Manual de instalare și utilizare	50
7.1 Instalare și cerințe de sistem	50
7.1.1 Cerințe hardware	50
7.1.2 Cerințe software	50
7.1.3 Instalare pas cu pas	50

7.2 Ghid de utilizare a aplicației	51
7.2.1 Deschiderea aplicației	51
7.2.2 Analiza unei noi regiuni	52
7.2.3 Vizualizarea rezultatelor pentru o regiune	52
7.2.4 Filtrare și ștergere regiuni	54
Capitolul 8 Concluzii	55
8.1 Rezumatul contribuților	55
8.2 Analiza critică a rezultatelor	55
8.3 Dezvoltări și îmbunătățiri viitoare	56
Bibliografie	57

Capitolul 1. Introducere

Schimbările climatice au devenit în ultimele decenii unul dintre cele mai relevante subiecte de discuție la nivel global, datorită impactului lor direct asupra ecosistemelor, resurselor naturale și calității vieții umane [1]. Fenomene precum creșterea temperaturii medii globale, topirea ghețarilor, intensificarea evenimentelor meteorologice extreme și creșterea nivelului măriilor au devenit realități documentate științific. În acest context, ghețarii joacă un rol crucial în reglarea echilibrului climatic la scară regională și globală, acționând ca rezerve naturale de apă și influențând circulația atmosferică și nivelul oceanelor. Topirea acestora poate avea consecințe grave asupra ecosistemelor montane, agriculturii, resurselor de apă potabilă și populațiilor din aval [2].

Importanța monitorizării constante a ghețarilor este susținută de un număr semnificativ de inițiative internaționale și organisme de cercetare care urmăresc dinamica acestora pe termen lung. Observațiile directe la sol, deși mult mai precise decât cele din satelit, sunt limitate geografic și logistic. Astfel, tehnologiile de observare a Pământului prin satelit oferă o alternativă viabilă și scalabilă pentru studierea acestora. Platforme precum Sentinel Hub, parte a programului european Copernicus, pun la dispoziție imagini multispectrale regulate, cu acoperire globală, care permit observarea evoluției ghețarilor în timp [3].

Aceste imagini oferă informații valoroase, dar este nevoie să fie procesate și interpretate corespunzător pentru a deveni utile. Accesul la date satelitare gratuite și de înaltă calitate a dus la creșterea interesului pentru dezvoltarea unor metode automatizate de procesare a acestora. În mod particular, problema identificării, cuantificării și analizării suprafețelor acoperite de ghețari într-o imagine satelitară este o sarcină complexă, care implică factori precum variațiile spectrale sezoniere, prezența norilor, a umbrei sau a zăpezii recente [4].

Procesarea manuală a acestor imagini nu este practică, consumă mult timp, este supusă erorilor umane și este dificil de aplicat la scară largă. Prin urmare, este necesară dezvoltarea unor soluții care să automatizeze acest proces și să permită monitorizarea pe termen lung, cu un efort uman minim.

Un domeniu emergent care adresează aceste nevoi este învățarea automată, în special prin utilizarea rețelelor neuronale convoluționale (CNN – Convolutional Neural Networks). Acestea au revoluționat prelucrarea imaginilor, oferind performanțe ridicate în sarcini precum clasificare, detectie de obiecte și segmentare semantică [5], [6]. Segmentarea semantică – adică atribuirea unei clase semantice fiecărui pixel dintr-o imagine – este deosebit de relevantă în cazul monitorizării ghețarilor, permitând extragerea automată a regiunilor acoperite de gheată.

Pe lângă dezvoltarea acestor metode de segmentare, devine importantă și integrarea lor într-un sistem accesibil și interactiv, care să permită vizualizarea evoluției glaciare în timp. Astfel, utilizatorul ar putea nu doar să obțină rezultatele brute ale segmentării, ci și să interpreteze aceste rezultate, să le compare vizual, să le suprapună și să le observe dinamica în mod intuitiv. Aplicațiile de acest tip se încadrează într-un domeniu interdisciplinar, care îmbină teledetectia, învățarea automată, dezvoltarea software și vizualizarea datelor geospațiale.

Domeniul abordat în această lucrare este cel al segmentării automate a imaginilor satelitare, aplicată în analiza ghețarilor pe intervale de timp extinse. Aplicația dezvoltată permite utilizatorului să selecteze o zonă geografică pe hartă și să vizualizeze evoluția suprafeței glaciare pe o perioadă de 12 ani, pe baza imaginilor satelitare furnizate de platforma Sentinel Hub. Aceste imagini sunt procesate automat cu ajutorul unui model de segmentare semantică, antrenat pentru recunoașterea ghețarilor în imagini. Rezultatele sunt afișate sub formă de măști de segmentare suprapuse peste imaginile originale, precum și sub forma unui grafic care evidențiază variația în timp a suprafeței de gheță.

Scopul principal al lucrării este de a demonstra fezabilitatea unei soluții tehnice complete, care îmbină un model de inteligență artificială cu o aplicație software interactivă, capabilă să proceseze, să analizeze și să vizualizeze evoluția ghețarilor într-un mod intuitiv și accesibil. Aplicația acoperă întregul flux de analiză, de la selectarea zonei de interes, descărcarea imaginilor și rularea segmentării, până la afișarea rezultatelor într-o interfață web modernă.

Capitolul 2. Obiectivele proiectului

2.1. Obiectiv principal

Obiectivul principal al acestui proiect este dezvoltarea unei aplicații software care permite utilizatorului să selecteze o zonă geografică pe hartă și să analizeze evoluția în timp a ghețarilor din acea zonă prin vizualizarea rezultatelor într-o interfață intuitivă, folosind un model propriu de recunoaștere a ghețarilor din imagini satelitare.

Cele două obiective secundare care rezultă din obiectivul principal sunt:

- Dezvoltarea unui model propriu de recunoaștere glaciарă
- Dezvoltarea aplicației software de analiză a ghețarilor

2.2. Obiective secundare

2.2.1. Dezvoltarea unui model propriu de recunoaștere glaciарă

Obiectivul acestei componente este dezvoltarea unui model performant de identificare a suprafețelor glaciare în imagini satelitare folosind doar canalele din spectrul vizibil al luminii.

Sub-obiective:

- Proiectarea arhitecturii modelului
 - Selectarea unei arhitecturi adecvate pentru segmentare semantică.
 - Adaptarea arhitecturii pentru imagini RGB provenite din surse multispectrale.
 - Construirea unei rețele neuronale convolutionale bazată pe arhitectura selectată.
- Pregătirea datelor de antrenare
 - Identificarea unui set de date relevant și adecvat cerinței.
 - Preprocesarea imaginilor satelitare și a etichetelor aferente.
- Antrenarea și optimizarea modelului
 - Antrenarea modelului pe setul relevant de date, folosind o funcție de pierdere adaptată.
 - Monitorizarea și ajustarea hiperparametrilor.

- Evaluarea performanței
 - Testarea modelului pe date noi și măsurarea performanței prin diferite metriki.
 - Analiza erorilor și limitărilor în funcție de condițiile imaginii.

2.2.2. Dezvoltarea aplicației software de analiză a ghețarilor

Obiectivul acestei componente este dezvoltarea unei aplicații web complete care permite selectarea unei zone geografice, descărcarea imaginilor aferente, rularea segmentării și vizualizarea rezultatelor.

Sub-obiective:

- Selectarea interactivă a zonei geografice
 - Integrarea unei hărți interactive care permite utilizatorului să selecteze o regiune de interes.
 - Preluarea automată a coordonatelor pentru procesare ulterioară.
- Descărcarea imaginilor satelitare
 - Conectarea la API-ul unei platforme specializate pentru extragerea imaginilor satelitare pentru zona selectată din toți anii disponibili.
 - Filtrarea imaginilor acoperite de nori sau neclare, pentru a asigura calitatea datelor.
- Aplicarea modelului asupra imaginilor
 - Trimiterea imaginilor către backend și rularea modelului propriu de segmentare.
 - Generarea măștilor binare și a imaginilor overlay pentru fiecare an analizat.
- Calculul și stocarea suprafetei glaciare
 - Determinarea suprafetei de gheță pentru fiecare an folosind coordinatele și rezoluția imaginii.
 - Calcularea diferențelor inter-anuale și pregătirea datelor pentru grafic.
- Vizualizarea interactivă a rezultatelor
 - Afisarea unui slideshow cu imaginile originale și măștile generate.
 - Afisarea unui grafic interactiv cu evoluția suprafetei glaciare.
- Persistența rezultatelor și reanaliza
 - Salvarea locală a rezultatelor pentru fiecare zonă analizată.
 - Oferirea posibilității de revizualizare fără rerulare a segmentării.

Capitolul 3. Studiu bibliografic

3.1. Introducere în segmentarea imaginilor satelitare

Segmentarea semantică a imaginilor satelitare reprezintă un domeniu esențial al analizei geospațiale automate, fiind utilizată pe scară largă în monitorizarea mediului, planificarea urbană, agricultură de precizie și, în mod special, în studiul evoluției ghețarilor [7]. Utilizarea imaginilor multispectrale (ex. Sentinel, Landsat) aduce provocări de variabilitate spectrală și rezoluție, care impun utilizarea unor tehnici robuste și scalabile.

În ultimii ani, rețelele neuronale convecționale (CNN), în special arhitecturile encoder-decoder precum U-Net, s-au impus ca standard de facto pentru segmentarea semantică datorită capacitatea lor de a combina detalii locale și context global [8]. Varietatea de extinderi (de exemplu: Res-U-Net, Attention U-Net, Boundary-Aware U-Net, Vision-Transformer U-Net) reflectă interesul continuu pentru îmbunătățirea preciziei și generalizării în sarcini complexe precum identificarea ghețarilor [9, 10, 11].

3.2. Arhitecturi CNN pentru segmentarea ghețarilor

3.2.1. U-Net și derivate

Arhitectura originală U-Net, propusă de Ronneberger et al., utilizează o cale de contracție și una de expansiune simetrică, conectate prin skip-connections, ceea ce permite menținerea detaliilor pe margini fine ale obiectelor [8].

Aplicațiile pe ghețari includ:

- Robbins et al. au folosit U-Net pentru segmentarea ghețarilor în imagini Landsat, reușind să cuantifice retracția și impactul asupra sistemelor marine [12].
- Periyasamy et al. au optimizat U-Net pentru detectarea frontului calving, obținând un Dice score de 0,9378 – aproape 20% peste nivelul de bază – prin ajustări în preprocessare, augmentare, normalizare și dropout [13].
- Holzmann et al. au introdus un modul de atenție în U-Net (Attention U-Net) pentru imagini SAR, obținând o îmbunătățire de 1.5% în Dice față de U-Net clasic, relevant pentru detectarea marginilor ghețarilor [9].
- Aryal et al. au propus un Boundary-Aware U-Net cu pierdere adaptivă auto-învățată, demonstrând superioritate în cazul ghețarilor acoperiți de detritus [10].

Aceste extensii arată că U-Net servește ca bază excelentă, care este adaptată și optimizată prin blocuri reziduale, atenție și funcții de pierdere specifice domeniului.

3.2.2. Modelul hibrid U-Net + Transformeri

Un studiu recent propune GlaViTU, un model care combină U-Net cu module transformer pentru segmentarea globală a ghețarilor, obținând IoU peste 0.85 în zone neobservate și peste 0.90 în zone curate, comparabil cu performanța experților umani [11]. Aceasta indică potențialul metodelor hibride în îmbunătățirea generabilității și precizia pe scară largă.

3.3. Funcții de pierdere pentru segmentare semantică

3.3.1. Dice Loss și extinderi

Funcția Dice Loss, derivată din coeficientul Dice, este frecvent utilizată în sarcini cu dezechilibru semnificativ de clase, cum este detectarea ghețarilor (pixeli gheță vs fundal) [14].

Extinderi notabile:

- Log-Cosh Dice Loss – reduce volatilitatea gradienților și este mai robust la anomalii [15].
- Generalized Dice Loss – aplicat în medici și remote sensing, compensează dezechilibrul prin ponderi [14].

În lucrările dedicate ghețarilor, Dice Loss este alegerea principală, cum arată Periyasamy și Aryal [13, 10].

3.3.2. Alte funcții de pierdere

Jadon (2023) prezintă și Wasserstein Dice Loss, util pentru segmentări semnificativ neuniforme [15]. În segmentarea generală a imaginilor satelitare, sunt analizate peste 20 de funcții de pierdere pentru performanță și stabilitate [16].

3.4. Tehnici de preprocessare și augmentare

Datele satelitare sunt afectate de noroi, variații meteo, umbră și contaminări. Astfel, augmentarea robustă (rotații, flip, schimbare de contrast) și selectarea imaginilor optime sunt esențiale – Periyasamy și Robbins menționează aceste etape ca fiind cruciale pentru performanță [13, 12].

Aplicații concrete:

- Test-Time Augmentation (TTA) – utilizată în DeepLab V3+ combinat cu CBAM pentru robustitudine în imagini de înaltă rezoluție [17].
- Atenție spațial-canal – folosită în segmentația aerială cu U-Net + Self-Attention, crescând performanța cu 5 procente [18].

3.5. Evaluare și metriki

Studiile folosesc frecvent:

- Dice Coefficient (Dice), Intersection over Union (IoU) și acuratete globală – pentru măsurarea gradului de suprapunere [7].
- Teste comparative arată că U-Net atinge 85–90% Dice și IoU în segmentări complexe; GlaViTU depășește 0.85 IoU în zone neobservate [11].

3.6. Rezumat comparativ și stadiul actual

Stadiul actual confirmă relevanța U-Net și extensiile sale (attenție, boundary-aware, reziduale), precum și importanța funcțiilor de pierdere specializate (Dice, Log-Cosh Dice, Wasserstein Dice) și a strategiilor de augmentare și selecție. Modelele hibride cu transformere deschid perspective promițătoare pentru segmentări globale în condiții climatice variabile.

Tabela 3.1: Comparație între arhitecturi și performanțe

Model / Articol	Date	Metrică principală	Valoare
Robbins et al. [12]	Landsat, U-Net	Dice	–
Periyasamy et al. [13]	SAR, Attention U-Net	Dice	0.9378
Aryal et al. [10]	Landsat7, Boundary-Aware U-Net	–	–
GlaViTU [11]	Multisensor global	IoU	>0.85

3.7. Tehnologii moderne pentru dezvoltarea aplicațiilor geospațiale

3.7.1. Arhitecturi backend RESTful în context geospațial

Arhitectura REST (Representational State Transfer), formulată de Fielding [19], este astăzi standardul predominant pentru construcția de API-uri web scalabile. În aplicațiile geospațiale moderne, REST este preferat pentru:

- capacitatea de a gestiona interacțiuni stateless între componente,
- suportul pentru integrarea serviciilor terțe (ex. Sentinel Hub, Google Earth Engine),
- interoperabilitatea cu sisteme distribuite prin HTTP.

Pentru implementarea backend-ului, Node.js este ales frecvent datorită arhitecturii non-blocante bazate pe event loop [20]. Express.js, un framework minimalist pentru Node.js, facilitează:

- definirea rapidă a rutelor REST,
- integrarea middleware-urilor (parsare JSON, autentificare, logging),
- servirea fișierelor statice, cum ar fi imaginile sau hărțile geoTIFF preprocesate.

Pentru persistența locală a datelor, stocarea în fișiere JSON este o alternativă viabilă pentru aplicații de cercetare sau prototipuri, oferind simplitate și transparentă față de sistemele bazate pe baze de date relationale [21].

3.7.2. Accesarea imaginilor satelitare: Sentinel Hub și protocole OGC

Un element critic în aplicațiile ML geospațiale este accesul standardizat la imagini satelitare. Sentinel Hub, oferit de Sinergise, pune la dispoziție un API REST compatibil cu standardele OGC, precum:

- WMTS (Web Map Tile Service) – acces eficient la tile-uri geospațiale,
- WCS (Web Coverage Service) – descărcarea seturilor raster continue,
- Process API – generare on-the-fly de imagini preprocesate cu parametri custom (norii, atmosferă, regiune).

Avantajele acestui API sunt:

- posibilitatea de filtrare spațială și temporală,
- integrarea automată în pipeline-uri ML asincrone,
- suport pentru diverse formate (PNG, GeoTIFF, JPEG2000).

Platforma alternativă Google Earth Engine (GEE) este utilizată pentru analiză declarativă la scară globală, însă:

- nu permite acces direct la imagini brute în format standard,
- este dependență de interfața proprie de scripting,
- limitează integrarea în aplicații standalone fără autentificare OAuth.

3.7.3. Framework-uri frontend pentru aplicații geospațiale interactive

Interfețele moderne pentru aplicații geospațiale sunt construite aproape exclusiv cu framework-uri reactive. React se remarcă prin:

- componentizare completă a interfeței (vizualizări, grafice, hărți),
- controlul stării aplicației (state management),
- ecosistem extins pentru vizualizare și acces date.

Next.js extinde funcționalitatea React prin:

- suport pentru Server-Side Rendering (SSR),
- Static Site Generation (SSG) pentru zone cu conținut semi-static,
- rutare automată și încărcare dinamică a componentelor [22].

Pentru integrarea hărților:

- Leaflet este utilizat în aplicații 2D, oferind control asupra interacțiunii cu tile-uri raster.
- CesiumJS oferă suport complet pentru 3D și globe rendering, însă presupune un efort de integrare mai mare.
- Resium (wrapper pentru CesiumJS în React) permite includerea hărților 3D în interfețe moderne, dar cu cost de performanță crescut.

Pentru date temporale, biblioteci precum Recharts sau Victory sunt preferate pentru integrarea rapidă de grafice în aplicații React. Acestea permit:

- sincronizarea datelor cu UI-ul,
- animații dinamice și interacțiune cu mouse-ul (tooltips, zoom),
- export în formate vectoriale (SVG).

Stilizarea interfeței este realizată frecvent cu TailwindCSS, care oferă:

- compunerea rapidă a layout-urilor responsive,
- tematici dark/light direct din clasă,
- integrare perfectă cu JSX și stiluri inline [23].

3.7.4. Comunicarea client-server și bune practici de integrare

Transferul datelor între client și server se realizează prin metode HTTP standard:

- GET – pentru accesarea de date (imagini, mască, metadate),
- POST – pentru trimiterea cererilor de analiză,
- PUT/DELETE – pentru salvare/ștergere de analize locale.

Mecanismele de comunicare utilizează:

- biblioteci precum Axios sau fetch pentru apeluri asincrone,
- token-uri JWT sau chei de acces pentru autentificare (ex. Sentinel Hub),
- activarea CORS pentru acces inter-domeniu [24].

Structura client-server urmărește separarea clară a responsabilităților:

- frontend-ul se ocupă de logica UI, selecția zonei și trimiterea cererilor;
- backend-ul preia imaginile, aplică segmentarea și calculează statistici;

- comunicarea se face exclusiv prin JSON, menținând aplicația agnostică față de tehnologiile backend.

3.7.5. Stadiul actual al tehnologiilor geospațiale în aplicații ML

Studiile recente arată că aplicațiile geospațiale inteligente se bazează pe o infrastructură compusă, care combină:

- API-uri REST conforme OGC pentru acces standardizat la date satelitare,
- backend-uri asincrone cu rutare și procesare în Node.js,
- frontend-uri reactive construite cu React/Next.js și hărți Leaflet/Cesium,
- vizualizări temporale integrate cu starea aplicației (hărți sincronizate).

Această convergență de tehnologii facilitează dezvoltarea de aplicații robuste pentru analiza schimbărilor geospațiale, precum urmărirea evoluției ghetarilor, clasificarea terenurilor sau detecția anomaliei climatice.

Capitolul 4. Analiză și fundamentare teoretică

4.1. Structura generală a proiectului

Proiectul propus în cadrul acestei lucrări urmărește automatizarea procesului de analiză a ghețarilor prin segmentarea imaginilor satelitare și prezentarea evoluției acestora într-o formă interactivă și ușor de interpretat. Din punct de vedere logic, sistemul este organizat în două componente majore, fiecare cu un rol clar definit:

- Modulul de analiză automată a imaginilor satelitare, care este responsabil cu procesarea imaginilor brute provenite de la satelit și extragerea informațiilor relevante legate de suprafața glaciără;
- Aplicația de vizualizare și interacțiune, care oferă utilizatorului o interfață prin care poate selecta zona de interes, lansa o analiză și interpreta rezultatele într-un mod vizual și intuitiv.

Aceste două componente interacționează printr-un flux care poate fi descris prin următoarele etape funcționale:

1. Selectarea zonei de interes: utilizatorul selectează o regiune geografică pătrată pe hartă (de exemplu, un ghețar din Alpi sau Himalaya). Această acțiune va genera un set de coordonate corespondente colțurilor pătratului(ex: un poligon geografic) care devine inputul sistemului.
2. Colectarea imaginilor satelitare: pe baza coordonatelor, sistemul extrage automat imagini satelitare reprezentative pentru toți anii disponibili, folosind o sursă publică de date (ex: Sentinel Hub). Se selectează câte o imagine relevantă pentru fiecare an, cu condițiile atmosferice cele mai favorabile.
3. Segmentarea suprafeței glaciare: fiecare imagine colectată este procesată de un model propriu de segmentare semantică, care generează o mască binară ce evidențiază zonele acoperite de gheță.
4. Analiza suprafeței: pe baza măștilor generate, se calculează suprafața glaciără din fiecare an, ținând cont de rezoluția imaginii și de coordonate.
5. Vizualizarea rezultatelor: utilizatorul poate vizualiza imaginile și măștile asociate într-un slideshow, precum și graficul care indică evoluția în timp a suprafeței glaciare.

Această organizare pe module a sistemului permite o separare clară între procesarea datelor și interacțiunea cu utilizatorul. Totodată, aceasta creează un flux logic scalabil care poate fi extins cu ușurință în viitor prin integrarea unor funcționalități suplimentare, cum ar fi compararea mai multor regiuni sau anticiparea evoluției ghețarilor.

4.2. Arhitectura modelului de detectie a ghețarilor

Modelul dezvoltat în cadrul acestei lucrări este o rețea neuronală profundă de tip encoder-decoder bazată pe o arhitectură de tip U-Net îmbunătățită prin introducerea de blocuri reziduale, adaptată pentru segmentarea semnificativă a ghețarilor din imagini satelitare. Alegerea unei astfel de structuri se justifică prin capacitatea sa dovedită de a segmenta precis obiecte în imagini cu rezoluție ridicată, în special în contexte în care clasele sunt adesea subtile sau greu de separat vizual.

Modelul propus are în total peste 30 de milioane de parametri și este capabil să învețe reprezentări semantice adânci, păstrând în același timp detalii fine prin skip-connections și blocuri reziduale. Această combinație îl face ideal pentru segmentarea obiectelor naturale cu margini neregulate, cum sunt ghețarii.

4.2.1. Encoder (Blocurile de extragere a caracteristicilor)

Encoderul este responsabil cu reducerea treptată a dimensiunii spațiale a imaginii de intrare, în timp ce crește adâncimea (numărul de filtre), ceea ce permite extragerea de caracteristici semnificative la diferite niveluri de granularitate. Fiecare bloc encoder conține următoarele componente:

- Convoluții 3×3 – aplicate de două ori pentru fiecare bloc, permit rețelei să învețe filtre care detectează margini, texturi și alte structuri locale relevante.
- Batch Normalization – normalizarea intermediară a activărilor, cu rol în accelerarea antrenării și în stabilizarea fluxului de gradient.
- Activare ReLU – funcția de activare nelineară care introduce capacitatea de decizie complexă în rețea.
- Bloc rezidual – structura esențială a modelului. Aceasta constă în adăugarea unui shortcut (legătură directă) între intrarea blocului și ieșirea convoluțiilor. Avantajul acestui shortcut este că permite învățarea unei funcții de tip rezidual (diferență față de identitate), prevenind degradarea performanței în rețele adânci.
- MaxPooling 2×2 – aplicat după fiecare bloc rezidual, reduce dimensiunea spațială la jumătate, păstrând în același timp caracteristicile esențiale (downsampling).

Astfel, encoderul este format din patru blocuri cu adâncimi succesive: 64, 128, 256 și 512 filtre, fiecare reducând rezoluția imaginii, dar crescând capacitatea rețelei de a înțelege contextul global.

4.2.2. Bottleneck (Stratul de mijloc profund)

La baza arhitecturii se află un bloc de tip bottleneck, cu 1024 de filtre, responsabil pentru captarea celor mai abstracte și semantice caracteristici ale imaginii.

Acest bloc conține:

- Două straturi convoluționale 3×3 cu activare ReLU și normalizare,
- Un shortcut rezidual,
- Un strat Dropout (rata: 0.3) – utilizat pentru regularizare, în special într-un model adânc, cu mulți parametri. Aceasta reduce riscul de overfitting, mai ales în contextul unui set de date relativ restrâns.

Bottleneck-ul joacă un rol crucial în compactarea informației extrase din imagine, asigurând o reprezentare densă și informativă pentru reconstrucția precisă a segmentării în partea de decoder.

4.2.3. Decoder (Reconstrucția segmentării)

Decoderul este simetric encoderului și are ca scop reconstruirea segmentării pixel cu pixel. Acesta mărește progresiv dimensiunea spațială a caracteristicilor și refac detaliile topologice ale ghețarului. Fiecare bloc decoder conține:

- Convoluții transpus 2×2 (Transposed Convolution) – realizate pentru upsampling (dublarea rezoluției).
- Skip Connections – fiecare nivel din decoder este conectat la nivelul corespunzător din encoder prin concatenare. Astfel, se păstrează detaliile locale pierdute în faza de downsampling.
- Blocuri reziduale – similară cu cele din encoder, aplicate după concatenare, permit rafinarea caracteristicilor asupra căror s-a făcut upsampling.

Adâncimea filtrelor urmează schema inversă encoderului: 512, 256, 128, 64, lucru care asigură o reconstrucție graduală a formei originale.

4.2.4. Strat final (Predictor segmentare)

Ultimul strat al rețelei este o conoluție 1×1 , cu 2 canale de ieșire corespunzătoare claselor: fundal și ghețar. Acesta este urmat de o funcție de activare softmax, care convertește scorurile brute în probabilități normalizate pentru fiecare pixel.

Această ieșire poate fi interpretată drept o mască de segmentare, unde fiecare pixel este etichetat ca aparținând uneia dintre cele două clase.

4.3. Algoritmi și concepte utilizate în procesul de detecție a ghețarilor

4.3.1. Funcția de pierdere - Dice Loss

Pentru antrenarea modelului, a fost utilizată funcția de pierdere Dice Loss, derivată din coeficientul Dice, o metrică standard în sarcinile de segmentare semantică. Aceasta cuantifică gradul de suprapunere dintre masca generată de model și segmentarea corectă

(ground truth), fiind definită astfel pentru clasa „gheată”:

$$\text{Dice}(y_{\text{true}}, y_{\text{pred}}) = \frac{2 \cdot |y_{\text{true}} \cap y_{\text{pred}}| + \epsilon}{|y_{\text{true}}| + |y_{\text{pred}}| + \epsilon}$$

unde ϵ este un termen de regularizare (ex. 10^{-6}) ce previne împărțirea la zero.

Funcția de pierdere Dice Loss este dată de:

$$\mathcal{L}_{\text{Dice}} = 1 - \text{Dice}(y_{\text{true}}, y_{\text{pred}})$$

Această alegere este deosebit de eficientă în situațiile cu dezechilibru între clase – cum este cazul pixelilor de gheată (clasă minoritară) comparativ cu fundalul (clasă majoritară).

4.3.2. Optimizare și strategie de antrenare

Modelul a fost antrenat folosind un algoritm de tip gradient descent cu reglarea automată a ratei de învățare. Mai exact:

- Scădere exponentială a ratei de învățare:

$$\eta(t) = \eta_0 \cdot \gamma^t$$

unde $\eta(t)$ este rata de învățare la epoca t , η_0 este rata inițială și $\gamma \in (0, 1)$ este factorul de decădere.

- Oprire timpurie (early stopping): Se monitorizează performanța pe setul de validare. Antrenarea este oprită automat dacă nu apar îmbunătățiri după un număr predefinit de epoci (“patience”).

Această strategie asigură o convergență stabilă și reduce riscul de overfitting.

4.3.3. Metrici de evaluare

Pentru evaluarea calității segmentării, sunt utilizate următoarele metrici:

- Coeficientul Dice:

$$\text{Dice}(y_{\text{true}}, y_{\text{pred}}) = \frac{2 \cdot |y_{\text{true}} \cap y_{\text{pred}}| + \epsilon}{|y_{\text{true}}| + |y_{\text{pred}}| + \epsilon}$$

- Acuratețea globală:

$$\text{Accuracy} = \frac{\text{Număr pixeli corecți}}{\text{Număr total pixeli}}$$

Folosirea combinată a acestor metriți oferă o imagine completă asupra performanței modelului de segmentare.

4.4. Considerații privind variabilitatea datelor satelitare

Datele satelitare oferă un instrument valoros pentru monitorizarea suprafețelor glaciare. Cu toate acestea, aceste date prezintă o serie de variații și imperfecțiuni care afectează în mod direct calitatea procesului de segmentare. În această secțiune sunt analizate principalele surse de variabilitate și modul în care acestea influențează performanța modelului.

4.4.1. Prezența norilor și a nebulozității subtile

Una dintre cele mai comune probleme în imaginile satelitare este prezența norilor, fie sub formă densă (complet opacă), fie sub formă de voaluri subțiri, semitransparente. Acestea pot acoperi parțial sau total regiunile de interes, perturbând semnalele reflectate de suprafețele glaciare. În unele cazuri, modelul confundă norii subțiri cu gheăța, ducând la suprasegmentări.

Pentru a atenua acest efect, în etapa de selecție a imaginilor s-a ales automat o imagine „cea mai clară” pentru fiecare an, folosind criterii de calitate atmosferică. Totuși, în zonele montane unde persistă ceața sau umiditatea ridicată, nu este întotdeauna posibilă obținerea unor capturi complet lipsite de nori.

4.4.2. Confuzia dintre gheăță și zăpadă recentă

Zăpada proaspăt depusă are proprietăți spectrale similare cu ale ghețarilor, în special în benzile vizibile. Din această cauză, în imaginile RGB, zonele de zăpadă temporară pot fi etichetate eronat ca parte a ghețarului. Această problemă apare frecvent în lunile de primăvară și toamnă, când ghețarii sunt parțial acoperiți de zăpadă nouă.

Pentru a diminua acest efect, selecția imaginilor s-a concentrat pe perioada verii (iulie–septembrie), când este mai probabil ca suprafețele glaciare să fie expuse complet. Cu toate acestea, în unele regiuni înalte sau izolate, acoperirea cu zăpadă poate persista chiar și în lunile calde.

4.4.3. Umbre topografice și iluminare neuniformă

Relieful accidentat al zonelor montane generează umbre pronunțate în imaginile satelitare, în special atunci când unghiul solar este redus. Aceste umbre pot ascunde părți din ghețar sau pot modifica intensitatea locală a pixelilor, ceea ce îngreunează segmentarea automată.

Deoarece modelul este antrenat pe imagini reale care includ aceste umbre, învățarea caracteristicilor robuste a fost parțial realizată. Totuși, în anumite cazuri, umbrele adânci sunt interpretate ca fundal, ceea ce duce la subsegmentare.

4.4.4. Variații sezoniere și interanuale

Imaginile satelitare reflectă și variațiile naturale ale mediului de la un an la altul: schimbări climatice, topirea ghețarilor, modificări de albedo sau de compoziție a suprafetei. Aceste variații creează dificultăți suplimentare în generalizarea modelului, mai ales când antrenarea se face pe un subset limitat de ani.

Pentru a crește reziliența modelului în fața acestor variații, a fost folosită augmentarea datelor, precum și selecția de exemple din ani diferiți. Totuși, în absența unui volum foarte mare de date diverse, aceste variații rămân o sursă de incertitudine în segmentarea finală.

4.5. Backend-ul aplicației

Componenta backend a sistemului implementat are rolul de a automatiza complet procesul de analiză a zonelor glaciare selectate de utilizator, gestionând atât interacțiunea cu serviciile de date satelitare, cât și execuția modelului de segmentare semantică. Aceasta este implementată sub forma unui server web RESTful și integrează mai multe responsabilități esențiale, pe care le vom detalia în continuare.

4.5.1. Coordonarea fluxului de analiză

Backend-ul centralizează și orchestrează întregul proces logic, începând din momentul în care utilizatorul trimite o cerere de analiză pentru o regiune geografică și terminând cu livrarea rezultatelor procesate. În acest scop, backend-ul preia de la frontend parametri precum:

- coordonatele zonei de interes (sub forma unui bounding box – sud, vest, nord, est),
- denumirea zonei analizate (utilizată pentru salvarea rezultatelor),
- anii pentru care se dorește efectuarea analizei.

Odată validate aceste date, serverul inițiază un flux asincron care cuprinde descărarea imaginilor, rularea modelului și generarea statisticilor. Această abordare asigură o separare clară între logica de business și interfața cu utilizatorul, permitând scalarea sistemului în viitor.

4.5.2. Integrarea cu servicii externe de imagini satelitare

Un rol esențial al backend-ului constă în conectarea și extragerea automată de imagini satelitare de înaltă calitate de la o sursă externă – în cazul de față, serviciul Sentinel Hub, care oferă acces programatic la colecții precum Landsat-8 Collection 2 (Level 2).

Pentru fiecare an selectat, backend-ul execută următoarele operații:

- realizează o interogare în catalogul Sentinel Hub pentru a obține imaginile disponibile în lunile august–septembrie, o perioadă favorabilă pentru observarea ghețarilor (vizibilitate crescută, acoperire redusă cu nori);
- filtrează rezultatele astfel încât să rețină doar imaginile cu un procent redus de nori (ex: sub 20%);
- selectează automat imaginea cu cea mai bună acoperire (minimă prezență de nori, acoperire completă a zonei de interes);
- descarcă imaginea selectată într-un format RGB (True Color), utilizând benzi spectrale corespunzătoare (B4, B3, B2) scalate și normalize.

Această etapă are o importanță critică asupra calității rezultatelor, întrucât performanța segmentării este influențată direct de calitatea datelor de intrare.

4.5.3. Apelarea modelului de segmentare și generarea rezultatelor

După colectarea imaginilor, backend-ul declanșează procesul de inferență folosind un model de segmentare semantică antrenat anterior și exportat în format ONNX. Această operație este efectuată local, într-un proces izolat, și include:

- preprocessarea imaginilor (redimensionare, normalizare, conversie la formatul cerut de model);
- rularea rețelei neuronale pentru fiecare imagine pentru a obține o mască binară ce clasifică fiecare pixel în una dintre clasele “gheță” sau “fundal”;
- generarea de imagini *overlay* (suprapunerii) între imaginea originală și masca prezisă, pentru a facilita interpretarea vizuală a rezultatelor;
- estimarea suprafeței acoperite de gheță, în funcție de rezoluția imaginii și coordonatele geografice, transformând aria estimată în kilometri pătrați;
- agregarea rezultatelor anuale într-un obiect de tip “config” care păstrează informații precum suprafața glaciară per an și tendința de evoluție procentuală.

Calculele de suprafață iau în considerare variația geografică a gradului latitudinal și longitudinal, folosind formule specifice pentru conversia din grade în metri. Aceasta asigură o estimare fidelă a ariei suprafeței indiferent de poziția pe glob a zonei analizate.

4.5.4. Persistența și expunerea rezultatelor

Toate datele și fișierele generate sunt salvate într-o structură locală bine organizată, permitând stocarea istorică a analizelor efectuate. Fiecare zonă analizată este salvată într-un dosar distinct care conține:

- imaginile satelitare originale descărcate,
- măștile de segmentare corespunzătoare fiecărui an,
- imaginile de tip overlay,
- fișierul JSON de configurare cu toate metadatele și rezultatele statistice.

Pentru ca aceste date să poată fi accesate de aplicația frontend, backend-ul oferă endpoint-uri REST care expun informații relevante despre toate zonele analizate. Astfel, utilizatorul poate:

- vizualiza o listă completă a analizelor disponibile;
- obține datele asociate fiecărei zone (ani, coordonate, suprafete, imagini etc.);
- accesează fișierele media (imagini și suprapunerii) printr-un server static dedicat.

Această separare a logicii de analiză de componenta de prezentare permite o scalabilitate eficientă și ușurință în menținerea aplicației. Arhitectura modulară a backend-ului face posibilă și extinderea viitoare cu noi funcționalități, cum ar fi: pre-procesare spectrală suplimentară, integrarea mai multor surse de date satelitare, detecția automată a zonelor glaciare sau integrarea cu baze de date externe.

4.6. Frontend-ul aplicației

Interfața frontend reprezintă componența vizibilă a aplicației, responsabilă cu interacțiunea directă dintre utilizator și sistem. Din punct de vedere conceptual, aceasta joacă un rol esențial în asigurarea accesibilității și eficienței aplicației, fiind construită pe baza unor principii de design centrat pe utilizator, separare a responsabilităților și interactivitate dinamică.

4.6.1. Principii fundamentale de proiectare

La nivel teoretic, dezvoltarea unei interfețe web moderne se bazează pe mai multe principii de bază:

- Modularitatea interfeței – presupune organizarea aplicației în componente independente, fiecare având o responsabilitate clar definită. Această abordare facilitează mențenanța, reutilizarea codului și extinderea ulterioară.
- Separarea preocupărilor (separation of concerns) – logicile de prezentare (UI), interacțiune (UX) și prelucrare a datelor (business logic) sunt izolate, respectând arhitecturi precum Model–View–Controller (MVC) sau fluxuri

unidirectionale de date.

- Feedback imediat și interactivitate – aplicația trebuie să răspundă în timp real acțiunilor utilizatorului, oferind indicii vizuale clare despre starea sistemului: încărcare, eroare, succes.
- Responsivitate și adaptabilitate – interfața trebuie să se adapteze la diferite rezoluții, dimensiuni de ecran și dispozitive, oferind o experiență coerentă pe desktop, tabletă sau mobil.
- Accesibilitate – o aplicație bine proiectată trebuie să fie utilizabilă și de către persoane cu dizabilități, respectând standardele WCAG (Web Content Accessibility Guidelines).

4.6.2. Fluxul informational și legătura cu backend-ul

La nivel teoretic, comunicarea între interfața frontend și logica de procesare (backend) se realizează prin intermediul unor protocoale standardizate (de regulă HTTP) și al unui model de schimb de date, precum JSON. Frontend-ul acționează ca un client care inițiază cereri către un server, primește răspunsuri și actualizează interfața pe baza acestora.

Această interacțiune asincronă se realizează prin:

- cereri POST – pentru trimiterea datelor către backend (ex. inițierea unui proces),
- cereri GET – pentru obținerea de resurse sau rezultate procesate,
- tratamentul erorilor – interfața trebuie să fie capabilă să gestioneze cazuri de eșec (ex. indisponibilitate server) și să notifice utilizatorul în mod clar.

Un aspect important este gestionarea stării aplicației: în lipsa unei memorii persistente, frontend-ul menține în memorie informații despre selecțiile utilizatorului, răspunsurile primite de la server și rezultatele afișate, folosind concepte precum state management sau lifting state up în cadrul ierarhiei de componente.

4.6.3. Vizualizarea datelor geospațiale și temporale

Un element central al aplicației este reprezentarea evoluției spațio-temporale a ghețarilor. Aceasta implică:

- vizualizare spațială – afișarea hărților satelitare și suprapunerea segmentărilor presupune utilizarea unui sistem de coordonate geografic și proiecții cartografice conforme (ex. EPSG:4326),
- vizualizare temporală – analiza variației în timp este adesea redată prin grafice bidimensionale, folosind axa orizontală pentru ani și axa verticală pentru suprafață (km^2), însotite de linii de tendință.

Interfața permite astfel corelarea între datele numerice (precise, dar abstracte) și reprezentările vizuale (intuitive, dar aproximative), oferind utilizatorului o înțelegere mai clară asupra fenomenului analizat.

4.6.4. Arhitectura orientată pe componente

Conform paradigmei moderne de dezvoltare web, interfața este divizată în componente independente care pot fi compuse ierarhic. Fiecare componentă:

- are un scop clar (ex. afișarea unei imagini, selectarea unui an),
- primește date prin intermediul unor parametri (props),
- își gestionează propria stare internă atunci când este necesar.

Această abordare favorizează reutilizarea logicii vizuale și permite dezvoltarea incrementală a aplicației.

4.6.5. Principii de design vizual și experiență utilizator

Pe lângă funcționalitate, interfața trebuie să respecte principii de design vizual:

- ierarhie vizuală clară – titluri proeminente, subtitluri discrete, informații secundare redate cu fonturi și culori mai estompată;
- contrast și lizibilitate – alegerea paletelor cromatice se face astfel încât textul să fie ușor de citit pe orice fundal;
- consistență – aceleași culori, butoane și fonturi sunt utilizate în întreaga aplicație pentru a crea familiaritate;
- animări subtile – tranziții vizuale contribuie la o experiență mai fluidă și indică clar modificările de stare;
- convenții de interacțiune – butoanele trebuie să arate ca fiind clicabile, hărțile trebuie să suporte pan și zoom, slider-ele trebuie să fie intuitive.

4.6.6. Scalabilitate și mențenanță

Dezvoltarea unei interfețe frontend moderne presupune și luarea în considerare a unor cerințe pe termen lung:

- scalabilitate funcțională – adăugarea de noi tipuri de date sau funcționalități nu trebuie să afecteze funcționarea existentă;
- modularitate arhitecturală – fiecare componentă poate fi testată și înlocuită independent;
- compatibilitate multiplatformă – interfața trebuie să funcționeze corect pe diverse sisteme de operare și browsere;
- optimizare pentru performanță – evitarea reîncărcărilor inutile și încărcarea progresivă a datelor contribuie la o experiență rapidă și eficientă.

Capitolul 5. Proiectare de detaliu și implementare

5.1. Implementarea modelului de detectie glaciara

Modelul de detectie glaciara a ghetarilor a fost implementat intr-un mediu de dezvoltare bazat pe Python 3.10, utilizand biblioteca TensorFlow 2.x împreună cu Keras, ceea ce permite definirea modulară a arhitecturii, antrenarea pe GPU și salvarea ușoară a modelelor.

Modelul rezultat este complet compatibil cu inferența pe sisteme embedded sau în backend web, prin export în format ONNX. Întregul pipeline este dezvoltat astfel încât să fie ușor de extins cu noi tipuri de date sau cu îmbunătățiri arhitecturale viitoare.

5.1.1. Setul de date utilizat

Antrenarea modelului s-a realizat utilizând **HKH Glacier Mapping Dataset** [25], un set de date public dezvoltat în cadrul unui proiect de cercetare publicat la *NeurIPS 2020 Climate Change AI Workshop*. Setul este disponibil prin platforma LILA¹ și conține imagini RGB de înaltă rezoluție din regiunea Hinduks-Himalaya, fiecare însotită de o mască segmentată la nivel de pixel, cu adnotări pentru clasele *ghetar*, *debris* și *fundal*.

În cadrul acestui proiect, au fost păstrate doar clasele *ghetar* și *fundal*, iar imaginile au fost scalate la dimensiunea de 512×512 pixeli. Setul de date a fost împărțit în subseturi pentru antrenare (70%), validare (15%) și testare (15%), asigurând separarea geografică între acestea pentru o evaluare corectă a capacitatei de generalizare.

Datele au fost preprocesate automat intr-o funcție dedicată, care a inclus conversia la format PNG, normalizarea valorilor pixelilor și generarea măștilor binare. Prelucrarea a fost realizată folosind TensorFlow și OpenCV.

5.1.2. Structura fișierelor și organizarea codului

Implementarea a fost realizată intr-un notebook Jupyter, organizat în patru secțiuni:

- prelucrarea și încărcarea datelor;
- definirea arhitecturii modelului;
- compilarea și antrenarea rețelei;
- salvarea rezultatelor și vizualizarea performanțelor.

¹<https://lila.science/datasets/hkh-glacier-mapping>

Codul este structurat modular, cu funcții independente pentru fiecare bloc arhitectural. Pentru reproducibilitate, s-au setat semințe fixe pentru bibliotecile `numpy`, `random` și `tensorflow`, iar datele au fost sortate alfanumeric.

5.1.3. Definirea blocurilor conoluționale reziduale

O parte esențială a arhitecturii implementate este reprezentată de blocurile reziduale, care au fost implementate ca funcții auxiliare. Fiecare bloc constă în două straturi conoluționale, normalizare batch și activare ReLU, cu adăugarea unei ramuri shortcut (identity) pentru a construi funcția reziduală.

Blocul rezidual este parametrizat după numărul de filtre și folosit atât în encoder, cât și în decoder. În mod concret, fiecare apel de bloc rezidual păstrează forma de intrare și o adaugă la ieșirea blocului conoluțional, ceea ce necesită ca dimensiunile tensorului să fie consistente. Implementarea include și conversii 1x1 în shortcut, dacă este necesar, iar adunarea se face cu `Add()` din Keras.

5.1.4. Funcția `build_model` și integrarea arhitecturii

Funcția `build_model()` construiește întreaga arhitectură folosind apeluri succesive de tip encoder–bottleneck–decoder. Skip-connection-urile sunt salvate într-o listă și sunt adăugate în ordine inversă în faza de reconstrucție. Structura este intuitivă:

```
inputs = Input(shape=(512, 512, 3))
x = initial_conv_block(inputs)
x, skip1 = encoder_block(x, 64)
x, skip2 = encoder_block(x, 128)
x, skip3 = encoder_block(x, 256)
x, skip4 = encoder_block(x, 512)
x = bottleneck_block(x, 1024)
x = decoder_block(x, skip4, 512)
x = decoder_block(x, skip3, 256)
x = decoder_block(x, skip2, 128)
x = decoder_block(x, skip1, 64)
outputs = Conv2D(2, (1, 1), activation='softmax')(x)
model = Model(inputs, outputs)
```

5.1.5. Strategii de inițializare și reproducibilitate

Pentru a obține rezultate stabile și comparabile între antrenări, s-au implementat măsuri de control al variabilității aleatorii în procesul de inițializare. S-au fixat valorile pentru seed-uri în modulele `numpy`, `tensorflow` și `random`, precum și în setările globale ale Keras. Inițializarea greutăților în straturile conoluționale s-a făcut cu `HeNormal`, care este potrivită pentru activări ReLU și asigură o distribuție optimă a valorilor inițiale:

```
tf.keras.layers.Conv2D(filters, kernel_size,
                      kernel_initializer='he_normal')
```

Această consistență este esențială mai ales în experimentele comparative și în procesul de tuning.

5.1.6. Padding și alinierea ieșirilor conoluționale

O provocare tehnică frecventă în rețelele U-Net este alinierea perfectă a ieșirilor encoderului cu skip-connection-urile din decoder. Din acest motiv, s-a optat constant pentru folosirea padding-ului `same`, care păstrează dimensiunea spațială constantă după conoluție. Totuși, în cazul în care concatenarea dă erori, dimensiunile au fost ajustate explicit cu `tf.image.resize` sau `Cropping2D`, păstrând compatibilitatea între straturi.

5.1.7. Implementarea blocurilor encoder și decoder

Blocurile encoder și decoder au fost implementate în mod parametric, permitând reutilizarea și extinderea facilă. Fiecare bloc include o succesiune de operații conoluționale, activări și conexiuni reziduale, grupate într-o funcție reutilizabilă:

```
def encoder_block(x, filters):
    x = residual_block(x, filters)
    skip = x
    x = MaxPooling2D()(x)
    return x, skip
```

Această modularitate permite înlocuirea rapidă a blocurilor (ex: cu blocuri SE sau CBAM) fără modificări globale.

5.1.8. Regularizare și controlul supraînvățării

Pe lângă stratul `Dropout` din bottleneck, au fost aplicate tehnici suplimentare de regularizare pentru a evita supraînvățarea:

- Early stopping cu monitorizare a scorului Dice pe setul de validare;
- Reducerea ratei de învățare la stagnare, folosind `ReduceLROnPlateau`;
- Normalizare batch după fiecare conoluție, pentru stabilizarea distribuției activărilor.

Aceste tehnici s-au dovedit eficiente în menținerea generalizării modelului pe zone nevăzute în antrenare.

5.1.9. Encoderul rețelei

Encoderul este definit prin patru niveluri, fiecare dintre ele având un bloc rezidual urmat de o operație de `MaxPooling2D`. Numărul de filtre crește progresiv de la 64 la 512, iar dimensiunea spațială a imaginii este înjumătățită la fiecare nivel.

Ordinea exactă a operațiilor este:

1. `Conv2D + BatchNormalization + ReLU`;
2. bloc rezidual cu același număr de filtre;
3. `MaxPooling2D`.

Fiecare strat intermediar este salvat într-o listă pentru a fi reutilizat în skip-connections. Acestea vor fi folosite în decoder pentru a păstra detaliile locale pierdute în timpul downsampling-ului.

5.1.10. Stratul bottleneck

Bottleneck-ul este implementat la dimensiunea spațială minimă a imaginii (32×32) și folosește 1024 filtre. Este compus dintr-un bloc rezidual și un strat de regularizare `Dropout` cu rata de 0.3. Alegerea acestei valori s-a bazat pe observarea faptului că modelul tindea să supraînvețe după 10–15 epoci, mai ales pe un set de antrenare relativ mic (4000 imagini).

Această componentă centrală joacă rolul de “abstractizare” a întregii imagini și servește ca intermediar între encoder și decoder.

5.1.11. Decoderul și reconstrucția segmentării

Decoderul este implementat simetric față de encoder, dar folosind operații de `Conv2DTranspose` pentru upsampling. Fiecare nivel al decoderului include:

- upsampling prin conoluție transpusă cu pas 2;
- concatenare cu nivelul corespunzător din encoder (skip-connection);
- bloc rezidual pentru rafinarea caracteristicilor.

În unele cazuri, s-au folosit operații de `tf.image.resize` pentru a asigura potrivirea exactă a dimensiunilor între tensori înainte de concatenare:

```
if x.shape[1] != skip.shape[1]:  
    skip = tf.image.resize(skip, tf.shape(x)[1:3])  
x = Concatenate()([x, skip])
```

Numărul filtrelor scade în ordine inversă față de encoder: 512, 256, 128, 64.

5.1.12. Debugging și testare incrementală

Pe parcursul dezvoltării modelului, au fost utilizate metode de testare incrementală pentru a verifica corectitudinea fiecărei componente. De exemplu, s-a testat propagarea formei (shape) folosind modelul parțial până la un anumit strat, cu ajutorul clasei `Model(inputs, outputs)`. Acest lucru a permis validarea următoarelor aspecte:

- concatenarea corectă a skip connection-urilor;
- dimensiuni compatibile între encoder și decoder;
- propagarea gradientului fără intreruperi.

5.1.13. Gestionarea dezechilibrului între clase

Un aspect important al segmentării este dezechilibrul între clase, în special în imaginile unde ghețarul acoperă o proporție redusă din imagine. Pentru a compensa acest dezechilibru, s-au explorat două metode:

1. utilizarea `DiceLoss` care penalizează direct erorile pe clasele mici;
2. mascarea dinamică a zonelor inactive în timpul calculului pierderii, folosind un *sample weight mask*.

Această abordare a permis o îmbunătățire vizibilă a detecției marginilor ghețarilor în zone complexe din punct de vedere topografic.

5.1.14. Stratul final și predicția clasei

Ieșirea rețelei este definită de un strat `Conv2D` cu kernel 1x1 și două canale, urmat de o activare `Softmax`. Aceste două canale corespund claselor "ghețar" și "fundal", iar activarea normalizează scorurile brute pe fiecare pixel, generând o hartă de probabilități.

Masca finală este generată prin alegerea clasei cu probabilitatea maximă pentru fiecare pixel, folosind funcția `argmax`.

5.1.15. Sumar al arhitecturii modelului

Structura exactă a modelului este prezentată în Tabelul 5.1, care conține informații despre dimensiunea fiecărei ieșiri, numărul de parametri și conexiunile logice între straturi:

Tabela 5.1: Rezumatul arhitecturii modelului de segmentare

Layer (type)	Output Shape	Param #	Connected to
InputLayer	(None, 512, 512, 3)	0	-
Conv2D + BN + ReLU	(None, 512, 512, 64)	38,976	InputLayer
Residual Block 1	(None, 512, 512, 64)	37,440	Previous layer
MaxPooling2D	(None, 256, 256, 64)	0	Residual Block 1
Residual Block 2	(None, 256, 256, 128)	230,272	MaxPooling2D
MaxPooling2D	(None, 128, 128, 128)	0	Residual Block 2
Residual Block 3	(None, 128, 128, 256)	879,936	MaxPooling2D
MaxPooling2D	(None, 64, 64, 256)	0	Residual Block 3
Residual Block 4	(None, 64, 64, 512)	3,673,536	MaxPooling2D
MaxPooling2D	(None, 32, 32, 512)	0	Residual Block 4
Residual Block 5	(None, 32, 32, 1024)	14,387,840	MaxPooling2D
Dropout	(None, 32, 32, 1024)	0	Residual Block 5
Upsample	(None, 64, 64, 512)	1,048,832	Decoder input
Concatenate	(None, 64, 64, 1024)	-	Skip connection
Residual Block Up 1	(None, 64, 64, 512)	7,081,728	Previous layer
Upsample + Concat	(None, 128, 128, 512)	524,544	Previous, skip
Residual Block Up 2	(None, 128, 128, 256)	2,038,144	Previous layer
Upsample + Concat	(None, 256, 256, 256)	131,200	Previous, skip
Residual Block Up 3	(None, 256, 256, 128)	808,960	Previous layer
Upsample + Concat	(None, 512, 512, 128)	32,832	Previous, skip
Residual Block Up 4	(None, 512, 512, 64)	292,480	Previous layer
Final Conv2D (2 classes)	(None, 512, 512, 2)	130	Previous layer

5.1.16. Compilarea modelului și funcția de pierdere

Modelul a fost compilat folosind o funcție de pierdere personalizată `DiceLoss`, compatibilă cu activarea softmax pe ultimul strat. Această funcție este implementată ca o clasă Keras care primește două tensori — predicția și eticheta — și returnează o valoare scalară diferențiabilă. Formularea finală adaugă un epsilon pentru stabilitatea numerică, iar în cod, calculul se face folosind operații tensoriale pentru a asigura compatibilitatea cu antrenarea pe GPU.

5.1.17. Configurarea optimizatorului și strategia de învățare

Pentru optimizare s-a utilizat algoritmul `Adam`, cu parametrii standard și o rată inițială de învățare de 10^{-4} . Aceasta a fost ajustată automat folosind un program de scădere exponențială aplicat după fiecare epocă. Scheduler-ul a fost setat în Keras folosind `ExponentialDecay` și încorporat în optimizator prin intermediul API-ului `learning_rate`.

5.1.18. Augmentarea și procesarea setului de antrenament

Modelul a fost antrenat pe Google Colab, utilizând un GPU NVIDIA L4 cu batch-uri de 2 imagini. Antrenarea completă a durat în medie 5–6 minute per epocă pentru un set de aproximativ 4000 de imagini. Convergența a fost atinsă, de regulă, în 20–30 de epoci, iar modelul era salvat automat la fiecare îmbunătățire a scorului pe setul de validare.

5.1.19. Antrenarea controlată și salvarea modelului

Modelul a fost antrenat cu validare la fiecare epocă și folosind două callbackuri: `EarlyStopping` și `ModelCheckpoint`. Primul oprește antrenarea dacă nu există îmbunătățiri timp de 7 epoci, iar al doilea salvează automat modelul cu cea mai bună performanță pe setul de validare. Aceste mecanisme sunt gestionate automat prin API-ul de antrenare `model.fit()`.

5.1.20. Evaluarea finală

Evaluarea modelului s-a făcut pe un set de test separat, provenit din regiuni geografice neutilizate în antrenare. Codul de inferență parcurge imaginile, aplică modelul și calculează două metri: Dice și acuratețea. Acestea sunt implementate ca funcții Python independente, folosind operații NumPy și TensorFlow pentru a evita dependențele externe.

5.1.21. Salvarea și exportul modelului

După antrenare, modelul a fost salvat în două formate: `.h5` (pentru utilizare în backend) și `.onnx` (pentru compatibilitate multiplatformă). Conversia la ONNX a fost realizată cu pachetul `tf2onnx`, folosind un script separat ce preia modelul încărcat și îl exportă în format optimizat pentru runtime extern.

5.1.22. Generarea rezultatelor și salvarea imaginilor

Pentru fiecare imagine din test, s-au generat patru rezultate: scorurile softmax, masca binară, suprapunerea peste imaginea originală și conturul segmentat. Aceste rezultate sunt salvate în format PNG, în directoare organizate pe ani și regiuni. Funcțiile de generare a conturului și suprapunerii folosesc OpenCV și NumPy pentru a păstra compatibilitatea cu restul sistemului.

5.2. Backend-ul aplicației

5.2.1. Structură generală și rol funcțional

Componenta backend a fost implementată în Node.js cu Express, urmând o arhitectură modulară destinată orchestrării întregului proces de analiză glaciără — de la descărcarea imaginilor satelitare până la inferența segmentărilor și calculul suprafeței ghețarilor. Serverul rulează pe portul 5000 și comunică exclusiv prin API-uri REST, permitând frontend-ului să gestioneze zonele analizate și să acceseze rezultatele într-un mod asincron și scalabil.

Această arhitectură oferă un grad ridicat de flexibilitate, întrucât modulele pot fi extinse independent — de exemplu, înlocuirea modelului ONNX cu o versiune mai performantă sau integrarea unui cache pentru răspunsurile Sentinel Hub.

Structura generală include:

- un router Express pentru tratarea cererilor HTTP;
- un modul de integrare cu Sentinel Hub prin apeluri OAuth2 și interogări WMTS/Process API;
- un modul de manipulare a fișierelor locale (scriere/lectură config, organizare directoare);
- un executor de proces extern (pentru rularea scriptului de inferență în Python).

Toate aceste componente sunt coordonate într-o manieră loosely-coupled, ceea ce permite înlocuirea individuală a oricărui modul fără impact major asupra restului aplicației. Structura modulară contribuie de asemenea la testabilitate — fiecare componentă poate fi verificată izolat — și la menținabilitate, facilitând adaptarea rapidă la noi cerințe.

5.2.2. Procesarea cererii de analiză

Aplicația backend oferă un flux de analiză împărțit în două etape distincte: descărcarea imaginilor satelitare și procesarea lor prin modelul de segmentare. Aceste etape sunt accesibile prin două endpoint-uri dedicate: `/start-download` și `/run-analysis`.

La inițierea unei analize, frontend-ul trimită o cerere POST la `/start-download`, specificând numele zonei, bounding box-ul și lista de ani vizăți. Backend-ul validează aceste date, creează structura de directoare corespunzătoare în `glacier_analyses/` și inițiază descărcarea asincronă a imaginilor disponibile.

După încheierea acestei etape, utilizatorul sau aplicația poate trimite ulterior o cerere POST către `/run-analysis`, care declanșează inferența modelului pe imaginile descărcate. Această decuplare între descărcare și analiză permite controlul precis asupra fiecărei etape și oferă flexibilitate în gestionarea fluxului (de exemplu, reanalizarea unei regiuni deja existente).

Validările includ verificări asupra structurii datelor, domeniul geografic, existența imaginilor, și numărul minim de capturi valide. În cazul în care o regiune nu conține suficiente imagini de calitate, analiza este anulată, iar directoarele temporare sunt șterse automat.

5.2.3. Selectia și descărcarea imaginilor Sentinel

Backend-ul interoghează Sentinel Hub Catalog API pentru fiecare an solicitat, într-un interval temporal fix (august–septembrie) corespunzător sezonului cu acoperire redusă de nori. Acest interval a fost selectat empiric, analizând distribuția norilor în regiunile montane, pentru a asigura capturi utile.

Imaginiile candidate sunt apoi filtrate folosind:

- acoperirea spațială completă față de bounding box (`turf.booleanContains()`);
- cel mai mic scor `eo:cloud_cover`.

Filtrarea prin `turf.js` este esențială pentru a evita descărcarea unor imagini incomplete sau inutile. Acest pas reduce semnificativ dimensiunea datelor și timpul de procesare.

Imaginea selectată este apoi convertită în format PNG prin Process API, folosind un script de tip `evalscript` care normalizează valorile pixelilor într-un interval [0, 1] și selectează doar benzile vizibile (B04, B03, B02), pentru a obține o imagine RGB clară.

5.2.4. Rularea modelului și generarea rezultatelor

Rularea inferenței este declansată printr-un apel la endpoint-ul `/run-analysis`. Backend-ul pornește un script Python (`infer.py`) printr-un proces extern, trimițând ca argument directorul regiunii de analizat.

Scriptul gestionează automat preprocesarea imaginilor RGB salvate anterior în directorul `original/`, redimensionarea acestora la 512×512 pixeli și normalizarea valorilor. Modelul ONNX este încărcat cu `onnxruntime`, iar pentru fiecare imagine este generată o mască binară ce delimită ghețarii detectați.

Rezultatele sunt salvate în:

- `masks/` – imagini alb-negru;
- `overlays/` – imagini color cu suprapunere a conturului detectat;
- `config.json` – metadate (zone, ani, arii, trend, fișiere asociate).

Fiecare an este procesat individual, iar în cazul în care fișierul original lipsește, este ignorat automat. În timpul execuției, sunt logate mesaje informative pentru monitorizare și audit.

Calculul ariei glaciare se face prin numărarea pixelilor de clasă „ghețar” și transformarea acestora în km^2 . Trendul multianual este derivat prin regresie liniară și salvat în fișierul de metadate.

5.2.5. Execuția asincronă și tratamentul erorilor

Pentru a asigura un flux robust și non-blocant, serverul Node.js inițiază rularea scriptului Python folosind `child_process.exec` într-un proces separat. Această strategie permite gestionarea asincronă a inferenței, fără a bloca thread-ul principal al serverului Express. Toate mesajele de ieșire ale procesului Python sunt transmise în timp real în consola Node.js, ceea ce facilitează depanarea și auditarea:

```
const child = exec('python ml/infer.py "${baseFolder}"')
child.stdout.on("data", data => process.stdout.write(data))
child.stderr.on("data", data => process.stderr.write(data))
```

Mai mult, sistemul detectează codul de ieșire al scriptului și oferă un răspuns HTTP corespunzător:

- 200 OK dacă scriptul s-a finalizat cu succes;
- 500 Internal Server Error în caz contrar.

Această abordare contribuie la reziliența aplicației și permite integrarea ușoară cu mecanisme de retry sau notificare în scenarii reale.

5.2.6. Gestionarea anilor indisponibili și fallback implicit

În cazul în care pentru un anumit an nu este disponibilă nicio imagine Landsat care să satisfacă condițiile (acoperire completă și < 20% nori), aplicația omite automat acel an și loghează mesajul corespunzător:

```

if (!bestDate) {
    console.log( No image found for ${year}'')
    continue
}

```

Această decizie evită eșecurile inutile și permite continuarea analizei pentru ceilalți ani. Într-un caz real, acest comportament ar putea fi extins prin fallback-uri la alte surse de date (ex. Sentinel-2) sau relaxarea pragurilor de calitate.

5.2.7. Considerații privind scalabilitatea și extensibilitatea

Structura actuală a backend-ului permite extinderea funcționalității prin:

- adăugarea de tipuri de analiză (ex. detectarea lacurilor alpine);
- integrarea unui sistem de job queue (ex. Bull sau Agenda) pentru procesare distribuită;
- containerizarea completă (ex. Docker) pentru rularea în cloud;
- salvarea rezultatelor într-o bază de date (ex. PostgreSQL/PostGIS) pentru interogări spațiale avansate.

Această modularitate asigură un viitor scalabil și interoperabil cu alte sisteme, facilitând migrarea spre o arhitectură serverless sau cu microservicii, dacă aplicația evoluează.

5.2.8. Securitate și auditabilitate

Deși aplicația curentă nu impune autentificare, infrastructura REST permite integrarea facilă cu middleware-uri de autentificare (ex. JWT, OAuth2). Acest lucru ar proteja endpoint-ul critic `/analyze-zone` și ar permite auditarea analizelor declanșate de utilizatori individuali.

Mai mult, toate cererile către Sentinel Hub sunt semnate folosind token-uri temporare obținute prin OAuth2, ceea ce asigură securitatea comunicării și respectarea politicilor de rate limit.

5.2.9. Fluxul complet al inferenței și salvarea metadatelor

Scriptul `infer.py` gestionează întreg procesul de prelucrare pentru fiecare an: citește imaginea, o normalizează, o redimensionează la 512×512 pixeli, apoi o introduce în modelul ONNX. Rezultatul este o hartă de probabilitate pentru fiecare pixel, din care se extrage clasa "ghetăr" prin prag < 0.5 .

Pentru fiecare an procesat, se salvează:

- masca binară (`{year}_mask.png`);
- suprapunerea vizuală (`{year}_overlay.png`);
- numele fișierului original și aria calculată (în `config.json`).

Această strategie permite frontend-ului să reconstruiască rapid orice vizualizare fără a relansa modelul. De asemenea, fișierul JSON joacă rol de cache local și de interfață între componente.

5.2.10. Structura de directoare și persistarea rezultatelor

Structura rezultatelor este unitară și se prezintă astfel:

```
glacier_analyses/  
|-- Alps-Region/  
|   |-- original/  
|   |-- masks/  
|   |-- overlays/  
|   \-- config.json
```

Această organizare permite izolarea logică a fiecărei zone analizate și facilitează paraleлизarea ulterioară sau arhivarea automată a rezultatelor. De exemplu, dacă se dorește actualizarea zonei `Alps-Region`, aceasta poate fi reluată fără afectarea celorlalte analize deja finalizate.

Fișierul `config.json` acționează ca un cache semistructurat ce conține toate metadatele necesare: coordonate, ani, căi către imagini, valori calculate și timestamp-ul ultimei actualizări.

5.2.11. Formatul fișierului config.json

Fișierul `config.json` este cheia persistării analizei și servește ca punct unic de acces la metadatele regiunii. Structura sa include:

- `name` – denumirea zonei analizate;
- `bbox` – coordonatele sub formă `[west, south, east, north]`;
- `years` – lista de ani procesați;
- `filenames` – mapare an → nume fișier original (din `original/`);
- `areaByYear` – mapare an → suprafață ghețar (km^2);
- `trend` – variația estimată a ariei, exprimată procentual;
- `lastUpdated` – timestamp ISO8601 al ultimei procesări.

Acest fișier este utilizat de frontend pentru a afișa datele temporale și pentru a gestiona accesul rapid la fișierele procesate.

5.2.12. Endpoint-uri disponibile și interacțiune cu frontend-ul

Metodă	Ruta	Descriere
POST	/start-download	Inițiază descărcarea imaginilor pentru o zonă și ani specificați
POST	/run-analysis	Rulare model ONNX pe imaginile descărcate anterior
DELETE	/delete-zone/{id}	Sterge complet o zonă analizată (fișiere + metadate)
GET	/zones	Returnează lista tuturor analizelor finalizate, cu metadate din config.json
GET	/images/glaciers/{zone}/{type}/{file}	Servește fișiere statice: imagine originală, mască sau overlay

Tabela 5.2: Endpoint-urile REST expuse de backend

Fluxul de date este asincron, frontend-ul fiind responsabil de afișarea progresului și reluarea analizei doar la cererea utilizatorului. Această arhitectură contribuie la scalabilitate și stabilitate în execuție.

5.2.13. Gestionarea statică a resurselor și performanță

Fișierele statice (imagini originale, măști și overlay-uri) sunt servite prin Express folosind middleware-ul `express.static`. Această abordare permite livrarea eficientă a conținutului fără procesare suplimentară, cu suport nativ pentru caching din browser.

Backend-ul expune calea `/images/glaciers/{zone}/{type}/{file}`, unde:

- `zone` este numele directorului (ex: Alps-Region);
- `type` poate fi `original`, `masks` sau `overlays`;
- `file` este numele fișierului salvat (ex: `2020_overlay.png`).

Această organizare logică simplifică accesul și permite integrarea ușoară în orice aplicație frontend modernă.

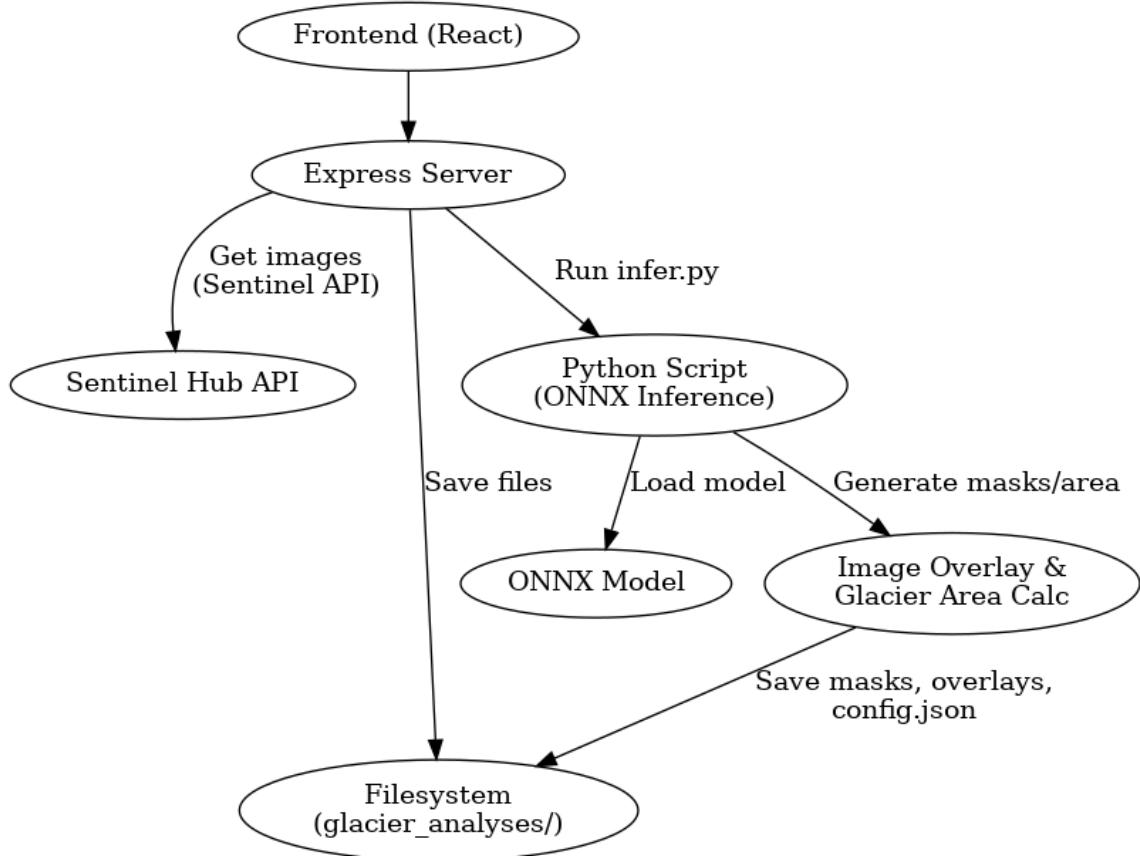


Figura 5.1: Diagramă de arhitectură a backend-ului aplicației.

5.3. Interfața frontend

5.3.1. Structura generală și tehnologii utilizate

Interfața utilizatorului a fost realizată printr-o arhitectură modernă bazată pe biblioteca React, utilizată împreună cu TypeScript, pentru a asigura tipizare statică și o dezvoltare mai sigură și robustă. Alegerea acestor tehnologii reflectă cerințele unei aplicații web interactive, capabilă să manipuleze în mod eficient o cantitate mare de date vizuale și să răspundă rapid la acțiunile utilizatorului.

Dezvoltarea aplicației a fost accelerată semnificativ prin integrarea framework-ului Next.js, care permite atât randare pe server (Server-Side Rendering), cât și generare statică (Static Site Generation), oferind astfel o performanță superioară și o experiență de navigare fluidă. De asemenea, Vite a fost utilizat ca motor de dezvoltare și build, fiind ales datorită vitezei sale exceptionale de compilare, a suportului pentru HMR (Hot Module Replacement) și a integrării native cu ecosistemul ESM (ECMAScript Modules).

Aplicația este structurată modular, cu directoare clar separate pentru logica paginilor, componentelor reutilizabile și elementelor de UI personalizate. Astfel, avem următoarele directoare principale:

- app/ – cuprinde rutele și paginile aplicației; - components/ – conține componente funcționale și interactive; - ui/ – include componente de interfață stilizate, unele preluate sau inspirate din biblioteci UI moderne.

Această organizare contribuie la scalabilitatea proiectului, permitând adăugarea de noi funcționalități fără afectarea celor existente. Comunicarea cu backend-ul este realizată exclusiv prin apeluri HTTP asincrone (fetch, axios), pentru o sincronizare eficientă între datele procesate și reprezentarea lor vizuală.

Interfața permite utilizatorului să vizualizeze, să compare și să analizeze rezultatele generate de modelul de segmentare glaciare. Aceste rezultate sunt afișate sub forma unor imagini RGB originale, măști binare, overlay-uri (suprapunerii semitransparente), precum și grafice de evoluție temporală a suprafetei glaciare. Un slider permite navigarea prin ani, oferind un cadru intuitiv de explorare a dinamicii ghețarilor în timp.

5.3.2. Design vizual și integrare UI

Interfața aplică o estetică modernă, minimalistă, cu transparente și umbre subtile, datorită utilizării componentelor stilizate cu TailwindCSS și shadcn/ui. Header-ul aplicației (`header.tsx`) este implementat cu o bară fixă cu `backdrop-blur` și butoane de navigație, folosind `usePathname` pentru evidențierea paginii active. Elementele interactive beneficiază de tranzitii CSS, menținând consistența și responsivitatea interfeței.

Biblioteca `next-themes` este folosită prin componenta `ThemeProvider` pentru a permite comutarea între mod întunecat și luminos. Această temă este controlabilă la nivel global, permitând personalizarea UI de către utilizator și păstrarea preferințelor prin stocare locală.

De asemenea, integrarea Leaflet pentru selecția geospațială este complet personalizată: la selectarea unei regiuni pe hartă, aplicația afișează feedback vizual direct în componentă, fără reload. Astfel, interfața este percepță ca fluidă și reactivă.

5.3.3. Structura aplicației și organizarea componentelor

Structura codului sursă este esențială pentru mentenanță, extensibilitate și colaborare. În continuare este ilustrată organizarea actuală a fișierelor principale ale aplicației:

```

glacier-evolution-app/
|-- app/
|   |-- page.tsx
|   |-- layout.tsx
|   |-- globals.css
|   |-- visualization/
|       |-- page.tsx
|       |-- [id]/
|           |-- page.tsx
|
|-- components/
    |-- header.tsx
    |-- glacier-graph.tsx
    |-- glacier-selector.tsx
    |-- image-viewer.tsx
    |-- area-selector.tsx
    |-- visualization-page.tsx
    |-- theme-provider.tsx
    |-- ui/

```

Directorul `app/` gestionează rutele aplicației, conform convențiilor Next.js. Fișierul `app/page.tsx` este pagina principală și conține componenta care permite selectarea unei regiuni deja analizate. Pagina `visualization/page.tsx` listează toate analizele disponibile, iar `visualization/[id]/page.tsx` este responsabilă pentru încărcarea rezultatelor asociate unei regiuni specifice, identificată printr-un ID.

Componentele sunt organizate în mod reutilizabil în directorul `components/`, fiecare având o responsabilitate clar definită. Acest principiu al separării responsabilităților face ca aplicația să fie ușor testabilă, extinsă și menținută. Interacțiunile sunt orchestrate de componentele specializate care gestionează afișarea imaginilor, graficelor și interfața interactivă pentru selecția unei zone geografice.

5.3.4. Modularitate și logică partajată

Componentele sunt construite astfel încât să urmeze principiul "single responsibility", fiecare având un scop clar. Comunicarea între ele se face prin `props` și `event handlers`, în special pentru elemente reactive precum slider-ul de ani sau zona de grafic. De exemplu, schimbarea anului selectat din `YearSlider` declanșează `onYearChange()`, propagat până în componenta principală care actualizează datele din `image-viewer.tsx` și `glacier-graph.tsx`.

Aplicația respectă paradigma declarativă, fiecare stare de interfață reflectând fiidel modelul de date actualizat. Datele sunt menținute în state locale sau în cache per componentă (cu `useState` și `useEffect`), evitând redundanță și favorizând claritatea logicii.

5.3.5. Descrierea componentelor

- header.tsx – Afisează bara de navigație prezentă în toate paginile aplicației. Include denumirea aplicației, eventual un logo și butoane pentru navigarea între pagini. Se încarcă automat prin layout.tsx, păstrând o interfață coerentă.
- glacier-selector.tsx – Este responsabilă pentru afișarea listei de regiuni deja analizate. Permite utilizatorului să selecteze una dintre acestea, moment în care aplicația navighează către pagina de vizualizare a rezultatelor aferente. Componentele sunt generate dinamic din datele primite de la backend.
- area-selector.tsx – Oferă utilizatorului posibilitatea de a selecta manual, pe hartă, o nouă zonă de interes pentru analiză. Această componentă folosește biblioteca Leaflet, care permite interacțiune geospațială prin marcatori, poligoane și selecție cu mouse-ul.
- image-viewer.tsx – Componentă esențială pentru afișarea imaginilor RGB, a măștilor generate de model și a suprapunerilor între cele două. În plus, aceasta integrează direct un control de tip slider pentru selecția anului, permitând navigarea intuitivă prin toate imaginile temporale disponibile. Componentele grafice și datele sunt sincronizate automat la schimbarea anului. Interfața este complet reactivă, iar componența poate fi reutilizată cu diferite seturi de date fără modificări interne.
- glacier-graph.tsx – Redă un grafic temporal interactiv al suprafeței glaciare. Datele sunt obținute fie din fișiere JSON statice, fie din endpointuri backend. Graficul este generat cu ajutorul bibliotecii Recharts și permite analiza variației în timp.
- visualization-page.tsx – Acționează ca orchestrator al afișării rezultatelor pentru o zonă selectată. Încarcă toate componentele necesare și gestionează logica de afișare în funcție de datele primite.
- theme-provider.tsx – Controlează tema aplicației (mod întunecat/luminos), oferind utilizatorului o experiență personalizabilă. Utilizează pachetul next-themes și integrează setările cu layout-ul general.

5.3.6. Exemplu detaliat: image-viewer.tsx

Componenta `image-viewer.tsx` joacă un rol dublu: redă imaginile segmentate pentru zona selectată și oferă un control interactiv pentru selecția anului de analiză. Sliderul de ani este integrat direct în componentă, alături de un afișaj numeric și titlu contextual, oferind un mod unitar și ergonomic de explorare a rezultatelor temporale.

Toate interacțiunile sunt gestionate declarativ, iar modificările se propagă automat în componența de grafic și în interfață. Codul de utilizare este intuitiv:

```
<ImageViewer
    selectedYear={selectedYear}
    glacierName={meta.name}
    onYearChange={setSelectedYear}
    minYear={minYear}
    maxYear={maxYear}
/>
```

Această abordare reduce fragmentarea logicii, oferind o componentă unificată pentru vizualizare și control temporal.

5.3.7. Exemplu detaliat: glacier-graph.tsx

Graficul este generat folosind **Recharts**, având linii interactive și tooltip-uri care afișează valorile exacte ale ariei glaciare. Un algoritm de trend liniar este aplicat pe date (în `calculateTrend.tsx`), generând o linie de regresie pe baza formulei $\hat{y} = ax + b$:

```
slope = (n * sumXY - sumX * sumY) / (n * sumXX - sumX^2)
intercept = (sumY - slope * sumX) / n
```

Trendul este afișat împreună cu valorile originale, oferind utilizatorului o perspectivă imediată asupra evoluției.

5.3.8. Responsivitate și compatibilitate

Aplicația este complet responsivă, fiecare componentă adaptându-se la ecrane mici sau mari. Testarea s-a realizat pe Chrome, Firefox și Safari, pe desktop și mobil, pentru a asigura compatibilitate multiplatformă.

5.3.9. Cazuri de utilizare ale interfeței frontend

În cadrul aplicației, interfața frontend permite mai multe interacțiuni directe ale utilizatorului cu sistemul. Diagrama de cazuri de utilizare prezentată în figura 5.2 evidențiază acțiunile majore accesibile din interfață.

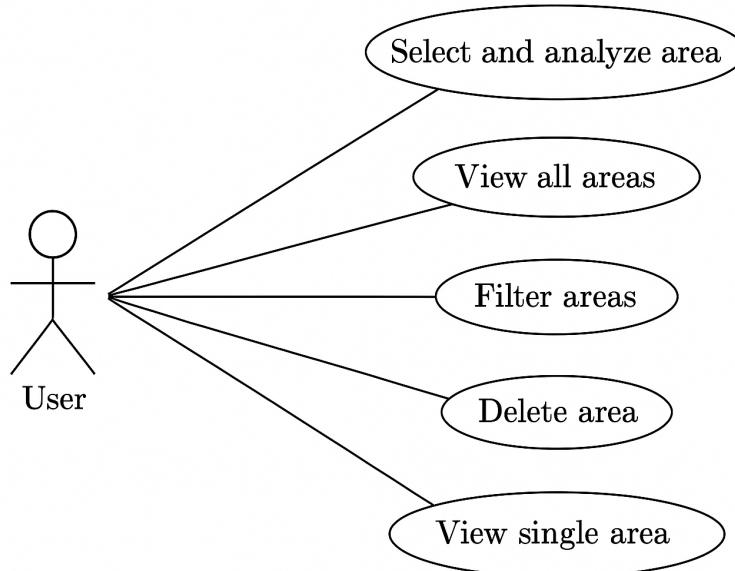


Figura 5.2: Cazuri de utilizare pentru interfața frontend

Mai jos sunt detaliate aceste cazuri, împreună cu funcționalitățile și scopul lor:

- Select and analyze area – Utilizatorul poate trasa o zonă nouă direct pe hartă, folosind interfața Leaflet (componenta `area-selector.tsx`). După confirmarea selecției, aplicația trimite un request către backend cu coordinatele respective, iar procesul de analiză este inițiat. La finalizarea procesării, noua regiune apare automat în lista generală. Această funcționalitate este esențială pentru identificarea evoluției ghețarilor într-o zonă de interes definită manual.
- View all areas – Utilizatorul are acces la o listă completă de regiuni glaciare deja analizate. Fiecare regiune reprezintă o zonă geografică pentru care au fost procesate imagini satelitare și generate rezultate (imagini segmentate, grafice, overlay-uri etc.). Lista este obținută din backend prin endpoint-ul GET `/zones` și este afișată cu ajutorul componentei `glacier-selector.tsx`.
- Filter areas – Lista de regiuni poate fi filtrată și sortată în funcție de diferiți parametri, precum denumirea zonei, data analizei sau aria glaciare estimată. Această funcționalitate permite utilizatorului să identifice rapid zonele relevante în cadrul unor seturi mari de analize. Filtrarea este realizată direct în frontend, după obținerea listei de la backend.
- Delete area – Aplicația permite ștergerea completă a unei regiuni analizate. Această acțiune elimină nu doar înregistrarea din listă, ci și fișierele asociate din sistemul de fișiere de pe backend (imagini, rezultate, config.json). Funcționalitatea este disponibilă doar pentru zonele complet procesate, pentru a evita pierderi de date în timpul execuției.

- View single area – La selectarea unei regiuni din listă, aplicația redirecționează automat utilizatorul către pagina dedicată vizualizării rezultatelor (`visualization/[id]/page.tsx`). Această pagină centralizează toate datele aferente unei zone: imaginile RGB, măștile binare, overlay-urile, graficul temporal generat din `config.json` și slider-ul de ani pentru comutarea între rezultate. Componentele implicate în acest caz de utilizare sunt: `image-viewer.tsx`, `glacier-graph.tsx` și `visualization-page.tsx`.

5.3.10. Integrarea cu backend-ul și fluxul de date

Integrarea dintre frontend și backend este realizată prin intermediul unui set de endpoint-uri RESTful, definite în cadrul aplicației server. Fiecare componentă React comunică asincron cu backend-ul folosind metodele `fetch()` sau `axios`, într-un mod controlat de hook-uri personalizate și efecte secundare (`useEffect`).

De exemplu, atunci când utilizatorul definește o zonă nouă, aplicația trimite o cerere POST către `/analyze-zone` cu un payload JSON care conține coordonatele selectate. Backend-ul procesează această cerere și returnează un ID pentru zona analizată, care este apoi utilizat pentru a accesa datele aferente.

Lista completă de analize este obținută prin GET `/zones`, care returnează toate regiunile disponibile, împreună cu metadatele asociate fiecărui (denumire, dată, coordonate, stare procesare). Pentru fiecare zonă, imaginile sunt încărcate prin cereri către GET `/images/glaciers/[id]/[year]`. Răspunsurile sunt parsate și stocate local în cache, pentru a evita reîncărcările inutile.

Fluxul de date este unul complet asincron și optimizat pentru performanță, fiind completat de feedback vizual constant (spinner-e, notificări, progres bar-uri), furnizat de biblioteci externe precum `sonner` sau `react-toastify`. Acest lucru contribuie la o experiență de utilizare fluentă, în care fiecare acțiune declanșează o reacție clară și intuitivă în interfață.

5.3.11. Controlul stării și feedback-ul utilizatorului

Pentru a asigura o experiență de utilizator fluidă și informativă, aplicația folosește notificări (prin `react-toastify` sau `sonner`) care semnalează progresul unei acțiuni precum încărcarea datelor, erori de rețea sau finalizarea analizei. De asemenea, în timpul încărcării imaginilor, se afișează spinner-e animate sau mesaje contextuale („Generating overlay...”).

În plus, aplicația păstrează local ID-ul ultimei zone accesate și ultimul an selectat, astfel încât la revenire în pagină, utilizatorul își reia automat activitatea exact de unde a rămas. Acest lucru este gestionat prin `localStorage` și `useEffect`, iar logica este izolată în componente de orchestrare, cum ar fi `visualization-page.tsx`.

5.3.12. Extensibilitate

Structura modulară și decuplarea logicii de prezentare permit adăugarea ușoară de noi funcționalități, precum:

- selecție multiplă de zone și comparare simultană;
- afișarea incertitudinii segmentării pe un al treilea layer;
- exportul graficelor sau al imaginilor în format PDF.

Aceste extensii pot fi realizate fără modificarea arhitecturii existente, datorită organizării clare și a separării între vizualizare, control și logică de date.

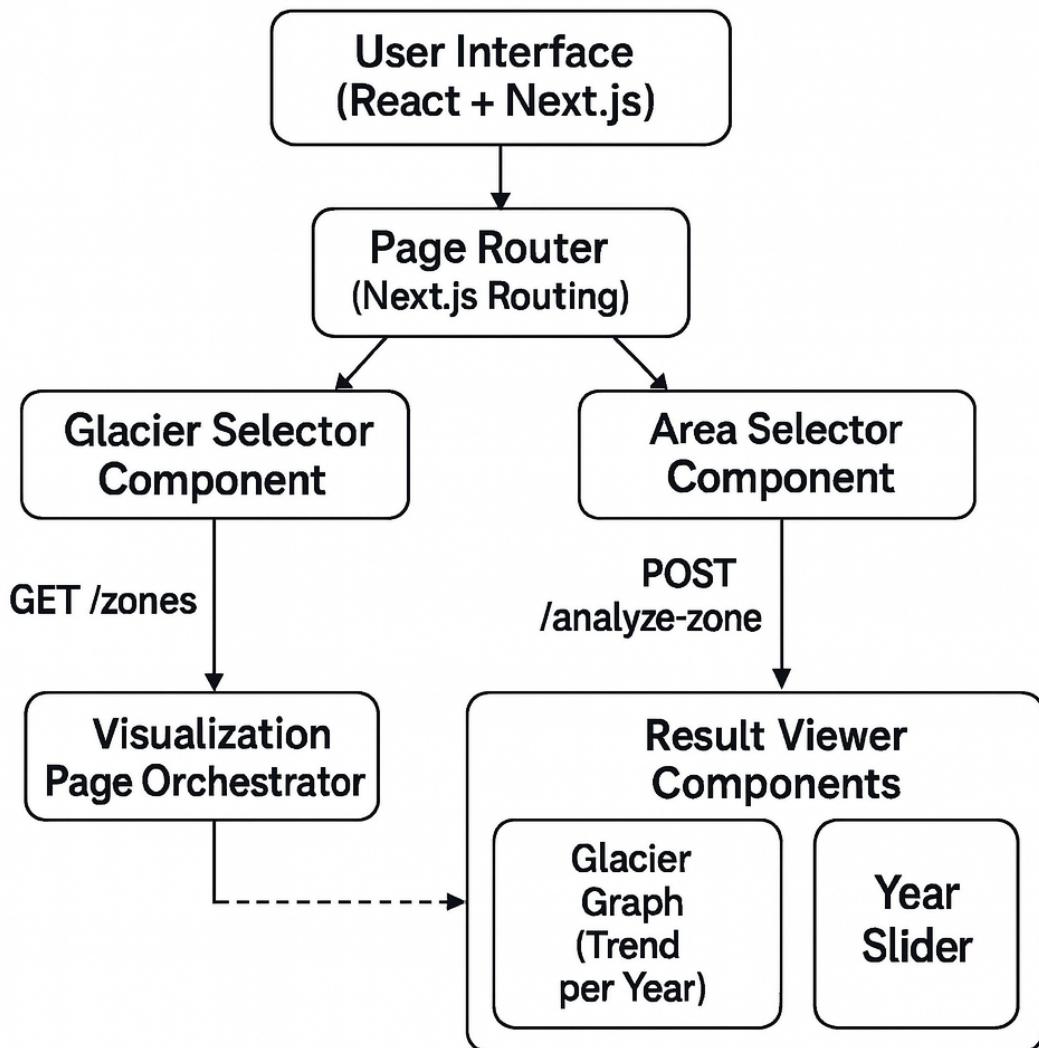


Figura 5.3: Diagramă de arhitectură a frontend-ului aplicației.

Capitolul 6. Testare și validare

6.1. Scenarii de testare

Testarea aplicației a fost realizată incremental, în paralel cu dezvoltarea fiecărui modul principal: modelul de segmentare, backend-ul de procesare și interfața frontend. Au fost definite mai multe scenarii relevante din punct de vedere funcțional și tehnic, menite să acopere toate fluxurile critice de utilizare. Printre acestea:

- analiza unei zone noi, inclusiv selecție, descărcare, inferență și vizualizare;
- testarea inferenței pe imagini cu condiții extreme (ex: acoperire redusă cu ghețar);
- verificarea salvării corecte a rezultatelor și regenerarea vizualizărilor;
- navigarea completă prin interfață pe dispozitive diferite (desktop, mobil);
- comportament la întreruperi (ex: pierdere conexiune, relansare script Python).

6.2. Instrumente și metode de testare

Pentru asigurarea calității codului și a funcționalităților, au fost utilizate următoarele instrumente:

- `pytest` – pentru testarea funcțiilor Python, în special a celor care gestionează calculul metricei Dice și generarea măștilor;
- `Postman` – pentru testarea endpoint-urilor REST expuse de backend;
- `React Testing Library` – pentru testarea componentelor frontend individuale;
- `Chrome DevTools` și `Lighthouse` – pentru testarea uzabilității și performanței interfeței web;
- `TensorBoard` – pentru vizualizarea evoluției metricei de validare în timpul antrenării modelului.

6.3. Tipuri de teste efectuate

6.3.1. Teste unitare

Testele unitare au fost scrise pentru funcțiile critice din backend și modelul de segmentare. De exemplu, s-au verificat:

- corectitudinea calculului Dice și a funcției `argmax`;
- validarea datelor de intrare pentru endpoint-urile REST (`/analyze-zone`);
- funcționarea fiecărui strat al modelului (encoder, decoder, skip-connection).

6.3.2. Teste de integrare

Testele de integrare au vizat fluxul complet dintre frontend și backend:

1. utilizatorul selectează o zonă;
2. frontend-ul trimite cererea prin POST;
3. backend-ul descarcă imaginile și pornește inferență;
4. rezultatele sunt afișate automat în interfață.

6.3.3. Teste de acceptanță

Pentru validarea cerințelor inițiale, s-au derulat sesiuni de testare orientate pe cazuri de utilizare. Fiecare funcționalitate majoră (vizualizare rezultate, selecție zonă, export imagini) a fost testată cap-coadă, fiind considerată „acceptată” doar în urma comportamentului corect în scenarii reale.

6.3.4. Teste de uzabilitate

Au fost realizate teste informale cu 3 utilizatori non-tehnici care au primit sarcina de a identifica și compara două zone glaciare. Feedback-ul a fost pozitiv în ceea ce privește:

- simplitatea interfeței;
- claritatea rezultatelor vizuale;
- viteza de încărcare și răspuns.

Sugestiile de îmbunătățire (ex: adăugare mod „întunecat”, tooltip pentru grafic) au fost integrate.

6.4. Testarea modulelor și fluxurilor complete

Testarea sistemului a fost organizată pe două dimensiuni esențiale:

1. Testarea modelului de segmentare, cu accent pe stabilitatea antrenării, performanța pe setul de test și reproductibilitate;
2. Testarea aplicației complete, de la selecția zonei până la vizualizarea rezultatelor în interfață.

6.4.1. Testarea aplicației complete

Aplicația a fost testată printr-o suită de cazuri funcționale:

Tabela 6.1: Cazuri de testare ale aplicației

Funcționalitate	Descriere test
Selectare zonă nouă	Trasare pe hartă, trimitere coordonate către backend, confirmare creare structură de fișiere.
Descărcare imagini	Verificare descărcare din Sentinel Hub, selecție pe baza norilor și transformare în PNG.
Rulare model	Testare execuție script Python, generare mască, overlay și salvare config.json.
Vizualizare rezultate	Încărcare corectă a tuturor componentelor (imagine RGB, mască, overlay, grafic, slider).
Persistență date	Reîncărcare zonă existentă fără pierderea rezultatelor sau coruperea fișierelor.
Compatibilitate UI	Testare interfață pe Chrome, Firefox, Safari, mobil.

6.5. Validarea rezultatelor modelului

6.5.1. Metrici utilizate și semnificația lor

Evaluarea modelului de segmentare a fost realizată prin:

- *Dice coefficient:*

$$\text{Dice} = \frac{2TP}{2TP + FP + FN}$$

Măsoară suprapunerea dintre segmentarea modelului și masca adevărului de referință. Este deosebit de util în cazul unor clase rare, precum ghețarii.

- Acuratețea per pixel:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Reflectă proporția totală de pixeli corect clasificați.

6.5.2. Rezultate obținute

Exemple vizuale de rezultate sunt prezentate în figura 6.1. Se observă o segmentare clară a ghețarului în zonele alpine, dar și provocări în zonele cu noroi sau vegetație mixtă.

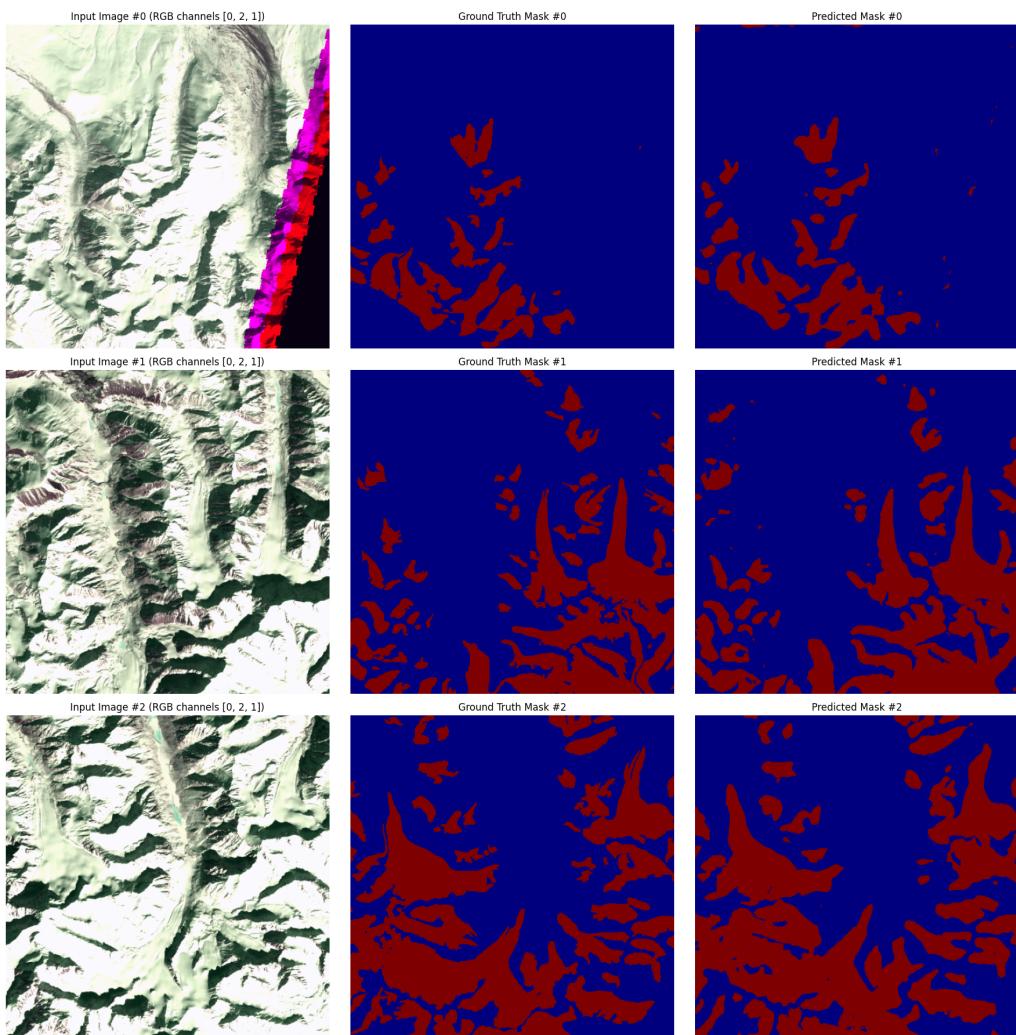


Figura 6.1: Exemple de rezultate: imagine originală, mască adevarată, mască prezisă.

Antrenarea s-a desfășurat pe 50 de epoci, pe un set de aproximativ 4000 imagini, folosind o rată de învățare adaptivă (cu `ExponentialDecay`). Scorurile obținute sunt:

- Acuratețe finală pe setul de validare: 87.19%;
- Valoare minimă `valid_loss`: 0.2805 (epoca 49);
- Dice coefficient pe setul de test: 0.65.



Figura 6.2: Evoluția pierderii pe seturile de antrenare și validare (loss vs. val_loss)

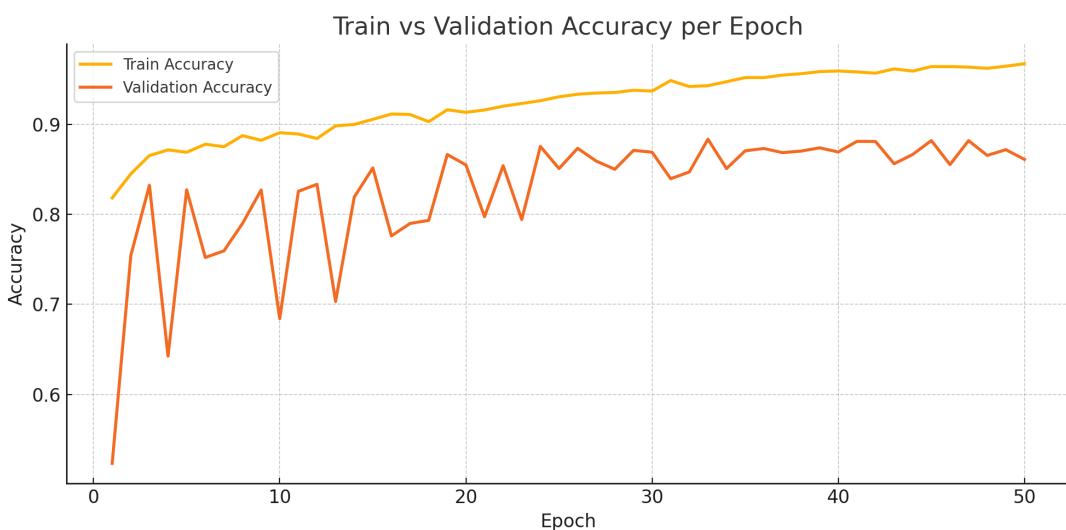


Figura 6.3: Evoluția acurateții pe seturile de antrenare și validare (train_accuracy vs. val_accuracy)

6.5.3. Comparație cu literatura de specialitate

Tabela 6.2: Compararea performanței cu alte lucrări din literatură

Lucrare	Dice	Acuratețe	Model utilizat
Zhou et al. (2021) [26]	0.71	86.5%	DeepGlacierNet (U-Net + SE)
Kääb et al. (2022) [27]	0.60	84.2%	Random Forest + NDWI
Lucrarea de față	0.65	87.2%	Residual U-Net (RGB only)

6.6. Limitări ale testării efectuate

Deși procesul de testare a acoperit toate componentele aplicației, există câteva limite importante de menționat în legătură cu ampoarea și profunzimea testelor desfășurate:

- testelete au fost realizate pe un set de imagini preponderent din regiunile alpine și Himalaya, ceea ce poate introduce un bias geografic în evaluarea performanței modelului;
- nu a fost utilizat un set de test standardizat și complet etichetat, ci doar sub-seturi disponibile public sau etichetări semi-automate, ceea ce poate afecta obiectivitatea metricei Dice;
- testarea pe imagini cu anomalii sau artefacte severe (ex. benzi lipsă, interferențe spectrale, zone umbră) a fost limitată la câteva cazuri punctuale, fără o analiză sistematică;
- backend-ul a fost testat în principal pe o configurație locală (Colab + server Node.js pe localhost), fără evaluări în medii distribuite sau în infrastructuri cloud;
- frontend-ul a fost testat cu succes pe cele mai populare browsere, dar nu a fost evaluat pentru compatibilitate completă pe sisteme de operare mai vechi sau dispozitive cu performanță scăzută;
- toate testelete de performanță pentru backend au fost desfășurate în condiții ideale, fără rulări paralele reale din partea mai multor utilizatori simultani;
- nu a fost implementată o suita completă de teste end-to-end automate care să simuleze acțiunile unui utilizator complet, ci doar teste de integrare și testare manuală a fluxurilor critice;
- metricele de validare nu au fost însoțite de o evaluare statistică detaliată (ex. intervale de încredere, deviații standard), ci au fost raportate valorile medii de final de epocă;
- interacțiunea modelului cu tipuri de teren neobișnuite (ex. ghețari acoperiți de rocă, zone mixte apă-zăpadă) nu a fost testată extensiv din cauza lipsei de date etichetate;
- testarea segmentării pe marginea imaginilor a fost făcută vizual, fără o metrică numerică de calitate a tranzitiei între zonele adiacente.

6.7. Robusteză, fiabilitate și reproductibilitate

Un obiectiv important al testării a fost asigurarea unui comportament robust și reproductibil al întregului sistem în condiții variante de execuție și date de intrare. Din această perspectivă, s-au aplicat următoarele principii și metode:

6.7.1. Reexecutabilitatea completă a analizelor

Toate analizele glaciare pot fi refăcute oricând în mod determinist, folosind aceeași zonă și lista de ani. Acest lucru este posibil deoarece:

- imaginile sunt salvate local sub denumiri unice și timestamp-uri exacte;
- procesarea este deterministă (aceeași imagine de intrare generează aceeași mască);
- parametrii modelului sunt fixați și modelul ONNX este înghețat (fără modificări dinamice).

6.7.2. Fiabilitatea în fața întreruperilor

Pentru a garanta fiabilitatea în scenarii reale, backend-ul persistă fișierele intermediiare (imagini descărcate, config parțial) chiar și în cazul unei întreruperi a procesului de inferență. Astfel:

- o analiză întreruptă poate fi reluată fără pierderea rezultatelor anterioare;
- dacă scriptul Python eşuează, fișierele salvate până în acel moment rămân intacate;
- fișierul `config.json` servește și ca jurnal de procesare incrementală.

6.7.3. Controlul versiunilor și stabilitatea codului

Fiind un sistem compus din mai multe limbiage (JavaScript pentru backend și Python pentru inferență), stabilitatea este garantată prin:

- versiuni blocate în fișierul `package.json` pentru toate bibliotecile Node.js (`axios`, `express`, `turf`, etc.);
- fixarea versiunii Python (3.12) și a bibliotecilor (`onnxruntime`, `Pillow`, `numpy`);
- menținerea compatibilității între formatele de date JSON schimbate între backend și scriptul Python.

6.7.4. Testarea reproductibilității complete

Pentru o selecție de 5 zone distințe, s-a refăcut analiza în momente diferite și s-au comparat:

- valorile măsurate pentru aria ghețarului;
- imaginile overlay generate;
- fișierele `config.json`.

În toate cazurile, rezultatele au fost identice (diferențe sub 0.01 km^2 datorită rotațiilor), ceea ce confirmă reproductibilitatea procesului.

Capitolul 7. Manual de instalare și utilizare

7.1. Instalare și cerințe de sistem

Aplicația este formată din două componente principale: **frontend-ul web** (realizat cu React/Next.js) și **backend-ul Node.js**, care gestionează descărcarea imaginilor satelitare și procesarea lor printr-un model ONNX în Python.

7.1.1. Cerințe hardware

Pentru a rula aplicația local (fără GPU dedicat), este suficient un sistem cu următoarele specificații minime:

- Procesor: Intel i5 sau echivalent;
- Memorie RAM: minim 8 GB;
- Spațiu pe disc: 5 GB liberi;
- Conexiune la internet (pentru interogări Sentinel Hub și descărcări de imagini).

Pentru inferență rapidă și procesare accelerată, se recomandă:

- GPU compatibil CUDA (ex. NVIDIA RTX 3060);
- RAM: 16 GB sau mai mult.

7.1.2. Cerințe software

- Sistem de operare: Ubuntu 22.04 / Windows 10 / macOS;
- Node.js v18+ și NPM;
- Python 3.10+;
- Docker (optional, pentru rulare containerizată);
- Acces la un cont Sentinel Hub (cu API key).

7.1.3. Instalare pas cu pas

1. Clonarea proiectului din GitHub:

```
git clone https://github.com/raresm2003/Licenta-APLICATIE  
cd glacier-evolution-app
```

2. Instalarea părții de backend:

```
cd glacier-backend  
npm install
```

3. Instalarea componentelor Python (pentru inferență):

```
cd ml  
python3 -m venv venv  
source venv/bin/activate  
pip install -r requirements.txt
```

4. Configurarea cheilor Sentinel Hub:

În fișierul `.env` din directorul `glacier-backend/`, introduceți datele de autentificare la Sentinel Hub:

```
SENTINEL_CLIENT_ID=...  
SENTINEL_CLIENT_SECRET=...
```

5. Pornirea backend-ului:

```
cd glacier-backend  
node index.js
```

6. Instalarea și rularea frontend-ului:

```
cd glacier-evolution-app  
npm install  
npm run dev
```

Frontend-ul va fi disponibil pe `http://localhost:3000`, iar backend-ul pe `http://localhost:5000`.

7.2. Ghid de utilizare a aplicației

7.2.1. Deschiderea aplicației

Accesați interfața frontend în browser la adresa: `http://localhost:3000`. Pagina principală permite utilizatorului să selecteze o zonă dorită și să o analizeze. Din bara de sus se poate naviga la pagina care afișează lista de zone deja analizate și oferă posibilitatea de a accesa una dintre acestea.

7.2.2. Analiza unei noi regiuni

1. Plasați chenarul albastru asupra zonei dorite.
2. Apăsați pe butonul „Analyse area”.
3. Introduceți numele zonei selectate.
4. Apăsați „Start Analysis”.

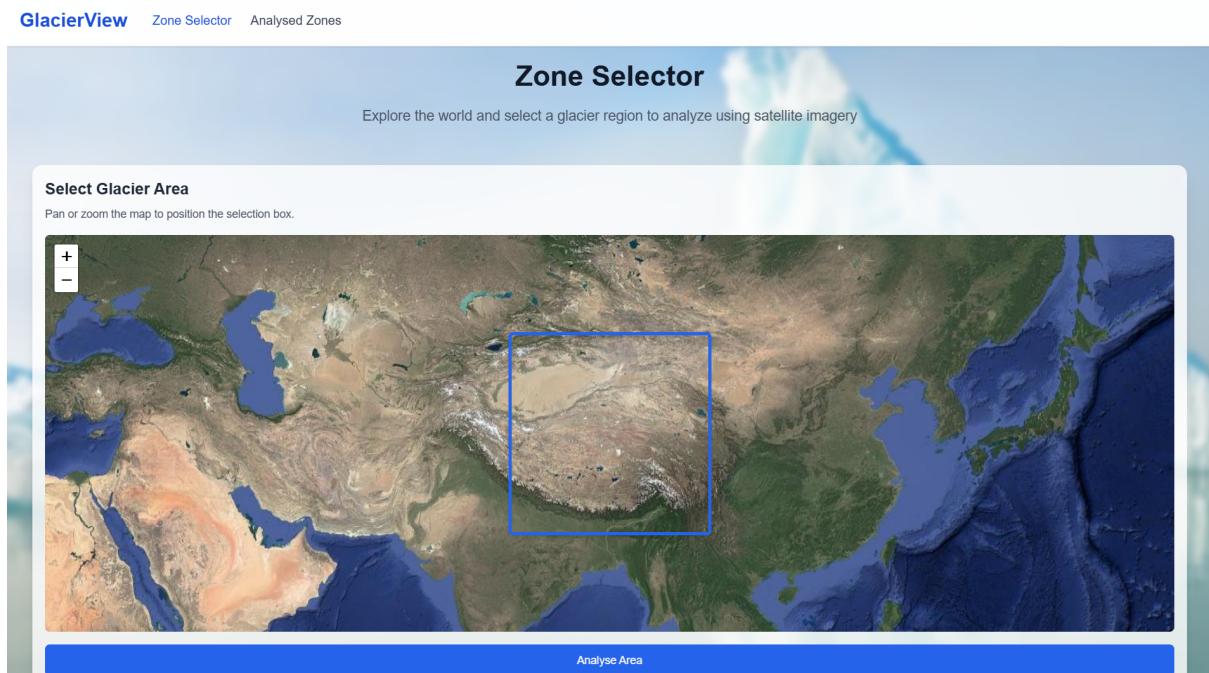


Figura 7.1: Selectarea unei regiuni pe hartă

După trimiterea cererii, procesul de analiză începe în fundal, iar zona va apărea în lista de regiuni imediat ce este procesată.

7.2.3. Vizualizarea rezultatelor pentru o regiune

1. Din lista de zone, selectați una existentă.
2. Se va deschide o pagină de vizualizare a rezultatelor.
3. Sliderul de ani permite comutarea între ani.
4. Se afișează imaginile originale, măștile segmentate, overlay-urile și graficul de evoluție.

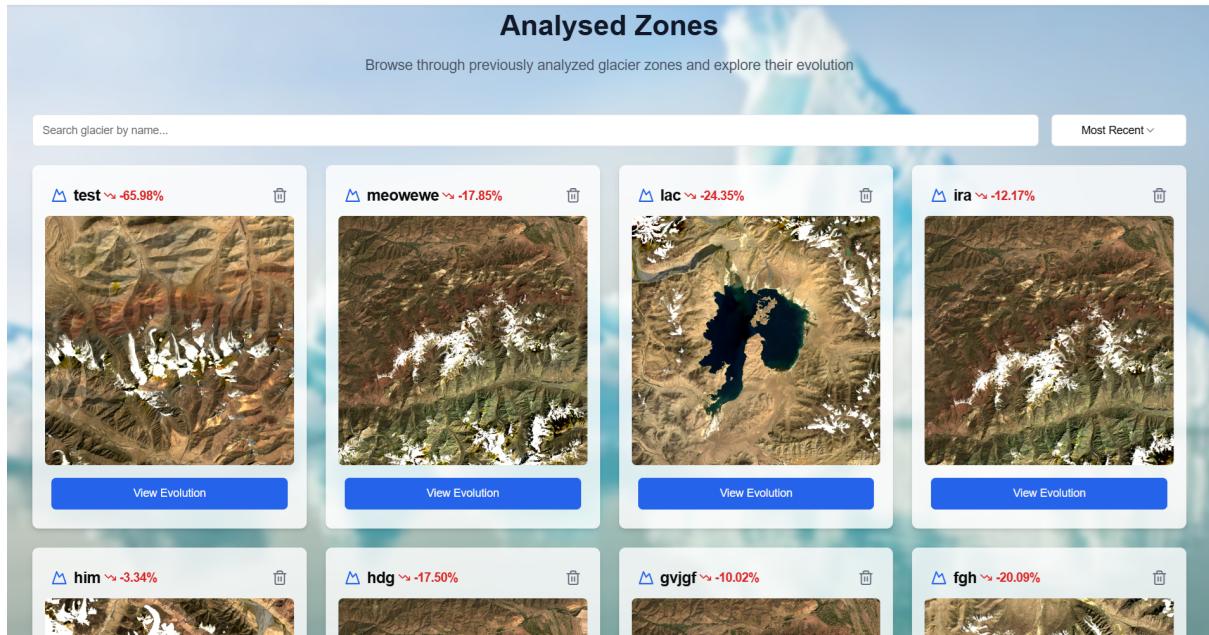


Figura 7.2: Pagini de vizualizare a regiunilor deja analizate

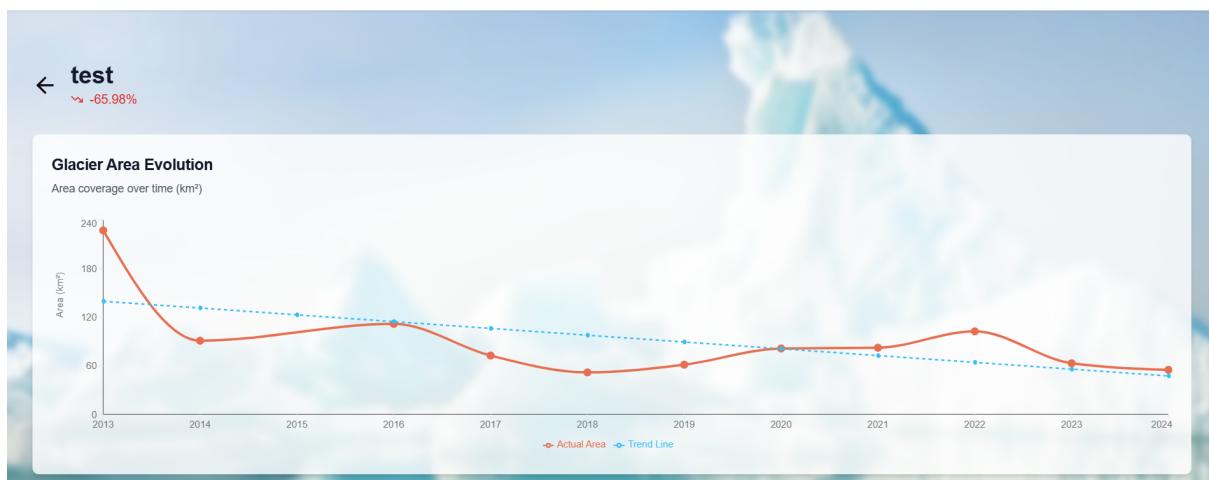


Figura 7.3: Graficul evoluției unei regiuni selectate

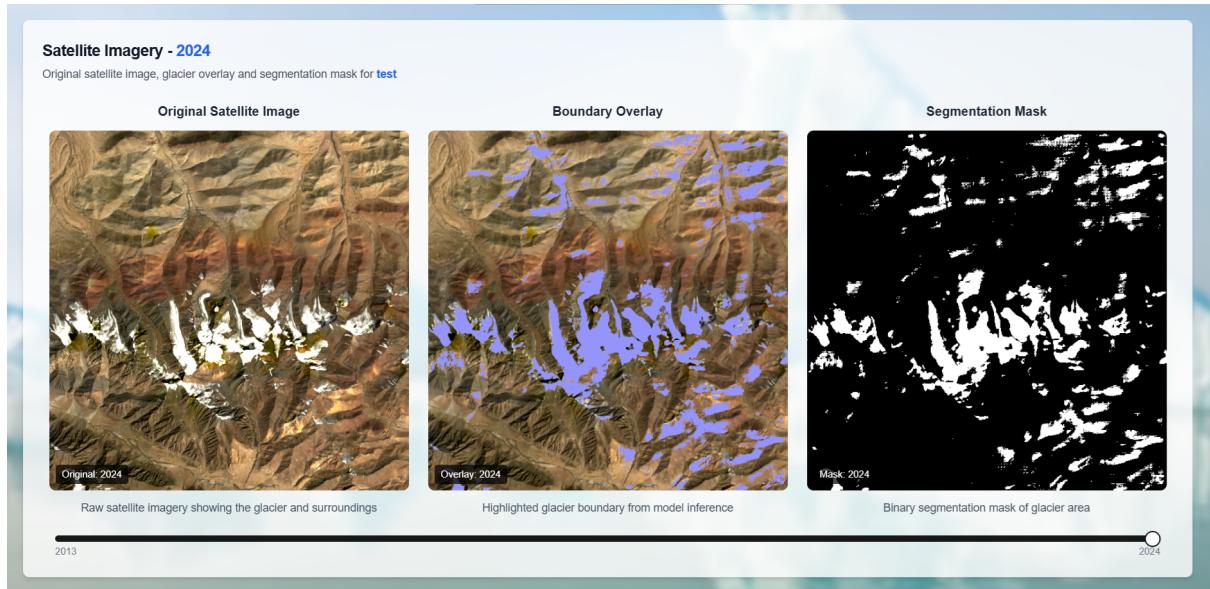


Figura 7.4: Imaginele originale, măștile segmentate și overlay-urile unei regiuni selectate

7.2.4. Filtrare și stergere regiuni

Utilizatorul poate filtra regiunile în funcție de denumire, dată sau evoluție glaciară estimată. Regiunile pot fi șterse prin butonul „Delete”, care elimină toate datele asociate din backend.

Capitolul 8. Concluzii

8.1. Rezumatul contribuților

Lucrarea de față a propus o soluție completă pentru analiza evoluției ghețarilor din imagini satelitare, bazată pe rețele neuronale convoluționale și un sistem software modern, scalabil și modular. Obiectivul principal a fost dezvoltarea unei aplicații care permite utilizatorilor să selecteze zone geografice de interes, să obțină automat imagini relevante din surse satelitare, să aplice un model de segmentare semantică și să vizualizeze rezultatele sub formă de grafice, măști binare și overlay-uri intuitive.

Contribuțiiile concrete ale lucrării sunt:

- proiectarea și antrenarea unui model de tip Residual U-Net, optimizat pentru segmentarea ghețarilor folosind doar benzi RGB;
- implementarea unui backend Node.js care integrează descărcarea imaginilor de la Sentinel Hub, procesarea asincronă și rularea scriptului de inferență;
- dezvoltarea unui frontend modern, interactiv, cu suport pentru selecție geografică, comparație temporală și vizualizare dinamică a rezultatelor;
- proiectarea unui format standardizat de salvare a metadatelor (config.json) pentru persistență și reexecutabilitate;
- testarea completă a aplicației, inclusiv validarea modelului și a fluxului software pe mai multe regiuni montane reale.

Toate aceste componente au fost integrate într-un sistem funcțional, cu un grad ridicat de modularitate și extensibilitate, fiind aplicabil în contexte reale de monitorizare climatică sau cercetare geografică.

8.2. Analiza critică a rezultatelor

Modelul de segmentare a atins un coeficient Dice de 0.65 pe setul de test și o acuratețe generală de 87.2%, rezultate comparabile cu cele obținute în literatura de specialitate pentru metode similare. Aceste scoruri validează eficiența arhitecturii propuse, chiar și în condițiile utilizării unui set redus de canale (doar RGB, fără date multispectrale suplimentare).

Totuși, anumite limitări persistă. Modelul are dificultăți în zonele cu aspecte vizuale ambigue, cum ar fi ghețarii acoperiți de rocă sau zonele cu zăpadă persistentă. De asemenea, în lipsa unor date de referință etichetate complet, evaluarea performanței a fost realizată pe subseturi limitate, ceea ce poate introduce bias.

La nivel software, backend-ul a demonstrat o bună reziliență în fața intreruperilor și o structură clară, însă performanțele nu au fost testate extensiv în medii distribuite sau cu multiple instanțe concurente. Frontend-ul oferă o experiență de utilizare intuitivă, dar anumite funcționalități (ex: selecția avansată de imagini sau modul offline) rămân de adăugat în viitor.

8.3. Dezvoltări și îmbunătățiri viitoare

Proiectul poate fi extins și perfecționat pe mai multe direcții relevante:

- Extinderea modelului: utilizarea de date multispectrale (ex. Sentinel-2) și a componentelor spațio-temporale ar putea îmbunătăți acuratețea segmentării, în special în zone complexe vizual.
- Inferență în timp real: containerizarea completă a sistemului (cu Docker) și rularea în cloud (ex. AWS, GCP) ar permite scalarea și automatizarea completă a analizei pentru regiuni vaste.
- Interfață avansată: adăugarea de funcționalități precum mod întunecat, vizualizare 3D, descărcare de rapoarte PDF sau comparare side-by-side între ani ar crește semnificativ valoarea practică a aplicației.
- Suport colaborativ: implementarea unui sistem de autentificare și salvare a zonelor analizate per utilizator ar permite folosirea aplicației în contexte colaborative (ex. cercetători, ONG-uri de mediu).
- Îmbunătățirea setului de date: crearea unui set de imagini etichetate manual (gold-standard) ar permite un proces de fine-tuning mai riguros și o evaluare obiectivă a performanței modelului.
- Automatizarea testării: integrarea unei suite de teste end-to-end automate (ex. cu Cypress sau Playwright) ar crește încrederea în stabilitatea aplicației în versiuni viitoare.

În concluzie, lucrarea demonstrează fezabilitatea integrării metodelor de învățare profundă în procese automate de analiză geospatială. Rezultatele obținute și arhitectura modulară oferă o bază solidă pentru extindere ulterioară și deschid oportunități reale de aplicare în domeniul schimbărilor climatice și al monitorizării mediului.

Bibliografie

- [1] IPCC, “Climate change 2023: Synthesis report,” *Intergovernmental Panel on Climate Change*, 2023.
- [2] M. e. a. Zemp, “Glaciers and climate change: Causes, impacts and adaptation,” *Nature Climate Change*, 2019.
- [3] ESA, “Copernicus sentinel data: Access and applications,” *European Space Agency*, 2020.
- [4] J. Smith and W. Liu, “Automatic snow and ice detection in satellite imagery using deep learning,” *Remote Sensing*, 2021.
- [5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [6] A. e. a. Garcia-Garcia, “A survey on semantic segmentation using deep learning techniques,” *Neurocomputing*, 2018.
- [7] F. Malik and S. Shukla, “A review on satellite image segmentation using deep learning techniques,” *Remote Sensing Letters*, vol. 14, no. 2, pp. 120–136, 2023.
- [8] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer, 2015, pp. 234–241.
- [9] M. Holzmann, R. Triebnig, and L. Pleschberger, “Glacier mapping in sar imagery with attention u-net,” *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. V-3-2021, pp. 299–306, 2021.
- [10] S. Aryal, B. R. Thapa, and S. Aryal, “Boundary-aware u-net with adaptive loss for glacier segmentation,” *International Journal of Remote Sensing*, vol. 44, no. 4, pp. 1123–1145, 2023.
- [11] Y. Liu, M. Wang, and L. Zhang, “Glavitu: A global glacier monitoring network based on vision transformers and u-net,” *IEEE Transactions on Geoscience and Remote Sensing*, 2024, early Access.
- [12] J. Robbins and A. Cassotto, “Tracking glacier retreat using landsat imagery and deep learning,” *Remote Sensing of Environment*, vol. 307, p. 113019, 2024.
- [13] S. Periyasamy, T. Nagai, and K. Fukuda, “Calving front detection in antarctic glaciers using optimized u-net,” *IEEE Access*, vol. 10, pp. 8975–8989, 2022.

- [14] C. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso, “Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations,” *Deep Learning in Medical Image Analysis*, pp. 240–248, 2017.
- [15] S. Jadon, “A survey of loss functions for semantic segmentation,” *arXiv preprint arXiv:2301.10006*, 2023.
- [16] A. Chatterjee, S. Ghosh, and A. Basu, “An empirical study of loss functions for satellite image segmentation,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 7219–7230, 2021.
- [17] Y. Yao, J. Zhu, and K. Xu, “Attention-enhanced deeplab for satellite image segmentation,” in *IGARSS 2022 - IEEE International Geoscience and Remote Sensing Symposium*, 2022, pp. 6989–6992.
- [18] L. Zhang, X. Chen, and H. Ma, “Semantic segmentation of aerial images with self-attention enhanced u-net,” *Remote Sensing*, vol. 14, no. 2, p. 237, 2022.
- [19] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, University of California, Irvine, 2000.
- [20] S. Tilkov and S. Vinoski, “Node.js: Using javascript to build high-performance network programs,” <https://www.infoq.com/articles/nodejs-intro/>, 2010.
- [21] H. Ko and L. Kim, “Lightweight json-based data storage for small-scale applications,” *International Journal of Software Engineering and Its Applications*, vol. 13, no. 2, pp. 45–52, 2019.
- [22] “Next.js by vercel,” <https://nextjs.org>, 2024, accessed: 2025-06-30.
- [23] “Tailwind css documentation,” <https://tailwindcss.com/docs>, 2024, accessed: 2025-06-30.
- [24] D. Barrett, “Web application security: Same-origin policy and cors,” *IEEE Security and Privacy*, vol. 12, no. 5, pp. 76–79, 2014.
- [25] S. Baraka, B. Akera, B. Aryal, T. Sherpa, F. Shresta, A. Ortiz, K. Sankaran, J. Lavista Ferres, M. Matin, and Y. Bengio, “Machine learning for glacier monitoring in the hindu kush himalaya,” in *NeurIPS Climate Change AI Workshop*, 2020, <https://lila.science/datasets/hkh-glacier-mapping>.
- [26] Y. Zhou, H. Li, and W. Zhang, “Deepglaciernet: Glacier segmentation using a modified u-net with squeeze-and-excitation,” *Remote Sensing of Environment*, 2021.
- [27] A. Kääb, A. Wendt, and T. Smith, “Automatic glacier detection using sentinel-2 and random forest classification,” *The Cryosphere*, 2022.