Universitatea POLITEHNICA din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Analiza semnalelor seismice folosind rețele neurale adânci

Proiect de Diplomă

Prezentat ca cerință parțială pentru obținerea titlului de *Inginer*
în domeniul *Calculatoare și Tehnologia Informației*
programul de studii *Ingineria Informației*

Conducător științific
Conf. dr. ing. Anamaria Rădoi

Absolvent
Alex-Costin Țițu

Anul 2023

Universitatea “Politehnica” din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației
Program de studiu **INF**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ

a studentului **TIŢU L. Alex-Costin, 443A**

1. Titlul temei: Analiza semnalelor seismice folosind rețele neurale adânci

2. Descrierea temei și a contribuției personale a studentului (în afara părții de documentare):
Semnalele seismice conțin o cantitate importantă de informație, ce poate fi extrasă folosind proceduri clasice ce permit analiza seismogramelor de manieră empirică sau folosind legi fizice. Proiectul curent adresează analiza semnalelor seismice din perspectiva extragerii informației pornind direct de la datele captate de senzori, cu ajutorul rețelelor neurale adânci și analizei timp-frecvență. Estimările parametrilor asociați semnalului seismic va fi realizată folosind date captate de la un singur senzor. În vederea validării metodei propuse, se va folosi ca măsură de performanță, eroarea medie pătratică. Pentru implementarea algoritmului, va fi utilizat limbajul de programare Python. Bibliotecile folosite sunt open-source (de exemplu, Numpy, Scipy, Pytorch).

3. Discipline necesare pt. proiect:
Recunoașterea formelor și inteligență artificială, Decizie și Estimare în Prelucrarea Informațiilor, Teoria transmisiunii informației, Semnale și sisteme, Structuri de date și algoritmi, Programare obiect orientată

4. Data înregistrării temei: 2022-12-05 18:44:43

Conducător(i) lucrare,
Conf. dr. ing. Anamaria RĂDOI

Student,
TIȚU L. Alex-Costin

Director departament,
Ș.L. dr. ing Bogdan FLOREA

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **48ff649f19**

Declarație de onestitate

Prin prezenta declar că lucrarea cu titlul „*Analiza semnalelor seismice folosind rețele neurale adânci*”, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității „Politehnica” din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul Calculatoare și Tehnologia Informației, programul de studiu *Ingineria Informației* este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinței bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 28 Iulie 2023

Absolvent Alex-Costin Țițu

.....

Cuprins

Lista figurilor

Lista tabelelor

Lista acronimelor

Adaptive Gradient (ro. Gradient Adaptiv) = AdaGrad

Artificial Neural Network (ro. Rețea Neurală Artificială) = ANN

Central Processing Unit (ro. Unitate Centrală de Procesare) = CPU

Convolutional Neural Networks (ro. Rețele Neurale Convoluționale) = CNN

Deep Learning (ro. Învățare profundă) = DL

Extremely Short Period - High Gain (ro. Perioadă Extrem de Scurtă – Amplificare Înaltă) = EH

Graphics Processing Unit (ro. Unitate Grafică de Procesoare) = GPU

High Broadband - High Gain (ro. Bandă largă – Amplificare Înaltă) = HH

Incorporated Research Institutions for Seismology (ro. Instituții de cercetare încorporate pentru seismologie) = IRIS

Inteligență Artificială = IA

Inteligență Computațională = IC

Mean Square Error (ro. Eroare Pătratică Medie) = MSE

Multilayer Perceptron (ro. Perceptron Multistrat) = MLP

Random-Access Memory (ro. Memorie cu Acces Aleator) = RAM

Raport Semnal Zgomot = RSZ

Residual Network (ro. Rețea Reziduală) = ResNet

Root Mean Square Propagation (ro. Propagarea Rădăcinii Pătrate Medii) = RMSProp

Short-time Fourier Transform (ro. Transformata Fourier de Timp Scurt) = STFT

STanford EArthquake Dataset = STEAD

Stochastic Gradient Descent (ro. Gradient Descendent Stocastic) = SGD

Transformata Fourier de Timp Discret = TFDT

Introducere

Avansurile tehnologice în domeniul arhitecturilor de calcul masiv paralele, facilitatea implementării și antrenării rețelelor neuronale adânci de complexitate ridicată cu ajutorul acestor arhitecturi și dorința constantă de a explora și înțelege activitatea și structura geologică a pământului au dus la un val recent de cercetări în domeniul interpretării semnalelor seismice.

Seismologia este o ramură a geologiei care se ocupă cu studiul științific al cutremurelor, create atât de surse naturale (mișcarea plăcilor tectonice, erupțiile vulcanice) cât și de surse artificiale (explozii subterane), și al propagării undelor prin scoarța terestră.[1] Una din sarcinile importante când vine vorba despre studiul seismologic al pământului este caracterizarea sursei seismice. Într-o situație de urgență, este dorită o diseminarea cât mai eficientă a informației, lucru care conduce la necesitatea unei estimări rapide și precise a caracteristicilor sursei seismice, preferabil fără intervenția unui expert uman. Cu toate acestea, majoritatea tehnicilor actual utilizate pentru caracterizarea surselor seismice presupun utilizarea datelor captate de la stații multiple, lucru care necesită utilizarea unor tehnici complexe de fuziune a datelor și specificarea și învățarea caracteristicilor geometrice ale rețelei.[2]

Cutremurele (sau seismele) sunt mișcări bruște ce apar în lungul faliilor, care eliberează energie elastică conținută în roci sub formă de unde seismice ce străbat Pământul. În fiecare zi au loc undeva la cincizeci de cutremure în toată lumea care sunt suficient de puternice (magnitudine > 2.5) să fie resimțite local, iar o dată la câteva zile are loc un cutremur suficient de capabil să deterioreze structuri. În plus, o multitudine de alte cutremure mici (magnitudine < 2.5) au loc, care deși sunt prea mici să fie resimțite, acestea pot fi înregistrate de către instrumentele moderne măsură. Aceste mici cutremure aduc informații foarte prețioase legate de procesul de formare al cutremurelor. [3]

Semnalele seismice conțin o cantitate semnificativă de informație ce poate fi extrasă folosind proceduri clasice de analiză a seismogramelor de manieră empirică sau folosind legi fizice. Lucrarea de față adresează analiza semnalelor seismice din perspectiva extragerii informației pornind direct de la datele captate de către senzori, cu ajutorul rețelelor neurale adânci și a analizei timp-frecvență. În acest fel vor putea fi estimați parametrii caracteristici ai surselor seismice înregistrate de către senzorul respectiv. Metrica de performanță a rețelei neurale în îndeplinirea sarcinii de estimare a parametrilor sursei seismice va fi eroarea (sau abaterea) pătratică medie.

Prin reducerea problemei la utilizarea doar a datelor ce aparțin unui singur senzor seismograf, dintr-o zonă geografică specifică, se vor putea extrage trăsăturile specifice zonei de care aparține prin analiza semnalelor seismice în domeniul timp-frecvență de către rețeaua neurală adâncă. Trăsăturile extrase vor fi utilizate pentru a estima parametrii cei mai importanți ai sursei seismice: magnitudinea, latitudinea, longitudinea și adâncimea cutremurului.

Alegerea unui singur aparat, ce aparține unei singure zone geografice specifice impune atât aspecte pozitive cât și negative asupra problemei.

Faptul că toate semnalele înregistrate vor aparține unei singure zone geografice specifice poziționării seismografului face ca problema învățării caracteristicilor structurii geologice a pământului din reprezentarea semnalului seismic în domeniul timp-frecvență să fie una mult mai facilă pentru rețeaua neurală. Majoritatea epicentrelor cutremurelor fiind poziționate într-o zonă geografică restrânsă, de obicei în apropierea faliilor, face ca fiecare seism să ai o caracteristică a formei de undă atât în domeniul timp cât și timp-frecvență specifică zonei din care provine. Astfel, semnalele înregistrate de către seismograf vor avea o formă de undă dată de specificul geologic al zonei din care provin cât și de calea străbătută de către unda de șoc până la senzor.

Dezavantajul major în utilizarea unui singur senzor seismograf în determinarea poziției exacte a hipocentrului (latitudine, longitudine și adâncime) este acela că informațiile legate de direcția din care semnalul provine sunt insuficiente pentru determinarea cu exactitate a poziției hipocentrului.

Pentru a determina poziția în două dimensiuni (de exemplu, într-un un plan – latitudine și longitudine), va fi nevoie de minimum trei senzori necoliniari. Măsurând diferențele de timp de sosire între senzori, se poate calcula diferența de distanțe dintre senzori și sursa semnalului. Aceste informații pot fi apoi utilizate pentru a triangula poziția sursei de semnal în două dimensiuni. Însă, pentru determinarea poziției în trei dimensiuni (de exemplu, în spațiul tridimensional – latitudine, longitudine și adâncime), este nevoie de minimum patru senzori necoplanari. Senzorul suplimentar permite estimarea poziției semnalului în a treia dimensiune, perpendicular pe planul format de ceilalți trei senzori.

Prin utilizarea unui singur senzor, problema va considerabil limitată, fiind posibilă determinarea cu o oarecare precizie a direcției din care semnalul a provenit și a distanței de la care acesta provine prin analiza caracteristicilor de intensitate ale undei în domeniul timp-frecvență. Acest lucru este puternic dependent de precizia măsurătorilor hipocentrilor seismelor etichetate în setul de date precum și complexitatea structurii geologice a zonei geografice respective.

Capacitatea ridicată de învățare a rețelelor neurale adânci împreună cu puterea de calcul disponibilă în zilele noastre, îmi vor permite implementarea a două tipuri de rețele convoluționale: o rețea convoluțională mai simplă, de tip LeNet și o alta mai complexă, cu o capacitate mai mare de extragere a trăsăturilor, de tip ResNet. Aceste rețele neurale vor analiza și extrage caracteristicile definitorii ale unei spectrograme aferente unei seismograme, iar aceste caracteristici vor fi utilizate în estimarea magnitudinii, epicentrului și adâncimii seismului.

Lucrarea urmărește determinarea capacităților rețelelor neurale adânci de tip convoluțional, implementate cu ajutorul bibliotecii PyTorch, de a extrage trăsăturile esențiale ale unui semnal seismic, captat de către un singur senzor, și e de a estima pe baza acestora caracteristicile principale ale sursei seismice.

Astfel, prin simplificarea ipotezei problemei doar la determinarea trăsăturilor unei singure zone geografice și utilizarea unor rețele neurale adânci cu potențial ridicat de extragere a trăsăturilor se va ținti către obținerea unor rezultate pozitive pentru analiza semnalelor seismice și estimarea tuturor parametrilor sursei de semnal. Singurul obstacol major existent va fi doar utilizarea unui singur senzor seismograf pentru estimarea coordonatelor hipocentrului într-un spațiu tridimensional.

Metoda propusă ce utilizează o rețea neurală de tip LeNet obține o eroare pătratică medie de 0.031 pentru estimarea magnitudinii, 0.034 pentru estimarea latitudinii, 0.042 pentru estimarea longitudinii și 4.19 km pentru estimarea adâncimii.

De asemenea, metoda propusă ce utilizează o rețea neurală de tip ResNet obține o eroare pătratică medie de 0.022 pentru estimarea magnitudinii, 0.022 pentru estimarea latitudinii, 0.050 pentru estimarea longitudinii și 3.84 km pentru estimarea adâncimii.

Rezultatele obținute dovedesc faptul că o rețea neurală de tip ResNet este, într-adevăr, mult mai performantă și mai potrivită pentru analiza semnalelor seismice comparativ cu o rețea de tip LeNet. Erorile pătratice medii obținute pentru latitudine și longitudine sunt promițătoare, însă insuficient de bune încât să poată fi declarate utilizabile în cadrul unor aplicații practice. Utilizarea datelor seismice ale unui senzor seismograf limitează semnificativ cunoștințele legate de seism, făcând dificilă estimarea cu precizie ridicată a latitudinii, longitudinii și adâncimii acestuia. Eroarea pătratică medie obținută în cazul magnitudinii, pe de altă parte, relevă faptul că performanța rețelei în cazul acestei sarcini este mult mai bună și ar putea estima acest parametru cu o precizie suficient de bună utilizând datele unui singur seismograf.

Capitolul 1. Fundamente teoretice ale Inteligenței Artificiale

1.1. Inteligența Artificială și Inteligența Computațională

Inteligența este abilitatea de a înțelege și a procesa o informație, de a folosi experiența anterioară și un set de informații apriori în vederea luării unei decizii viitoare informate pe baza acestora. Inteligența Artificială (en. Artificial Intelligence) reprezintă capacitatea mașinilor de a reproduce acest tip de inteligență pe baza unui set de informații de intrare, de a aviza situațiile sau deciziile viitoare. Inteligența Artificială mai poate fi definită și ca domeniul de studiul ce urmărește proiectarea agenților inteligenți, unde un agent inteligent este un sistem care percepe mediul înconjurător și ia decizii pentru maximizarea șanselor de succes.[4]

Inteligența Computațională este un subset al Inteligenței Artificiale, care se definește ca o alternativă a domeniului inteligenței artificiale convenționale și se bazează pe algoritmi euristici de inspirație naturală.[4] Domeniul este specific focalizat pe învățarea unei mașini, crearea unui algoritm care să învețe din experiențe, din date, fără a fi explicit programat cum să proceseze acel set de informații de intrare.

Totul a plecat de la încercarea de a reproduce capacitatea creierului uman de a procesa informațiile foarte rapid, cu o mare precizie și de a putea fi antrenat sau poate chiar să învețe singur să identifice forme incomplet descrise. Astfel au luat naștere rețelele neuronale artificiale (Artificial Neural Networks-ANN) ca o încercare de modelare a sistemului nervos, devenite obiect matematic. Rețelele neuronale oferă o nouă paradigmă de calcul diferită de cea a calculatoarelor actuale prin aceea că nu execută programe secvențiale de rezolvare algoritmică a problemelor, ci se instruiesc sistematic prin utilizarea unor eșantioane de învățare. Informația este distribuită redundant în structura rețelei, iar modelul de calcul este puternic paralel și distribuit.[4]

Debutul domeniului rețelelor neuronale artificiale își are locul în anul 1943, cu primul model de neuron propus de către W. S. McCulloc şi W. Pitts, model acceptat în mare parte și în zilele noastre. Modelul de neuron propus era un dispozitiv simplu ce realiza suma ponderată a intrărilor, ponderile pozitive fiind ponderi excitatoare, iar ponderile negative fiind inhibitoare. Dacă excitația totală (suma ponderată a intrărilor) depășește un anumit prag, atunci neuronul este activat și emite un semnal de ieșire (y = 1), altfel neuronul nu este activat (y = 0). Modelul de neuron propus de McCulloc şi Pitts este cunoscut şi sub numele de “poarta liniară cu prag” (Linear Threshold Gate).[4]

Progresele înregistrate în neurobiologie și psihologie au dus la apariția unor modele matematice ale învățării. Un astfel de model a fost propus de D. O. Hebb în anul 1949, acesta explicând modul în care ponderile sinaptice sunt ajustate pentru a reflecta mecanismul de învățare realizat de neuronii interconectați atunci când aceștia sunt influențați de stimuli ai mediului.[4]

În anul 1958, Frank Rosenblatt a publicat prima sa lucrare despre perceptron. Această lucrare prezenta perceptronul ca “un model probabilistic pentru memorarea și organizarea informației în creier”. Perceptronul este format prin interconectarea unei mulțimi de neuroni artificiali și reprezintă primul model de rețea neuronală artificială.[4]

1.2. Modelul Neuronului Artificial

Neuronul sau perceptronul este blocul structural fundamental al unei rețele neuronale. Ideea principală a unui neuron este de a primii un set de informații de intrare, iar pe baza acestora să facă o estimare a valorii dorite la ieșire.

Un *neuron artificial* (en. Artificial Neuron) este un set de patru entități:

$$\nu = \{X, W, \theta, f_{AN}\},$$

caracterizat prin implementarea unei transformări neliniare

$$f_{AN}: R_n \rightarrow [0, 1]$$

sau

$$f_{AN}: R_n \rightarrow [-1, 1],$$

unde $X = (x_1, x_2, \ldots x_n)^t$ este intrarea neuronului artificial reprezentată ca un vector real cu n componente, $W = (w_1, w_2, \ldots w_n)^t$ este vectorul n-dimensional reprezentând setul de ponderi al neuronului, θ este un parametru real numit pragul neuronului artificial, iar y este ieșirea acestuia.

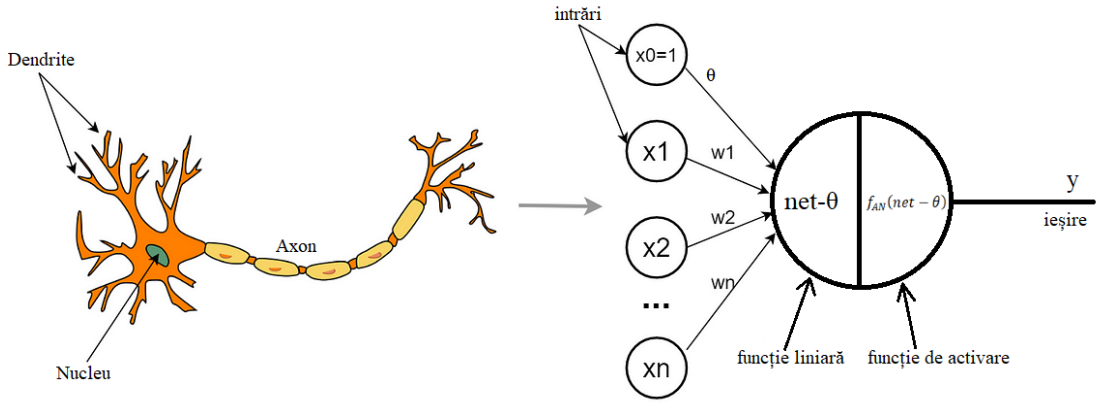


Figura 1.1 Neuronul artificial cu n intrări $x_1, x_2, \ldots x_n$, ponderile $w_1, w_2, \ldots w_n$, pragul θ și cu ieșirea y; Sursa: [5]

Intrarea netă a neuronului este calculată uzual ca suma ponderată a tuturor semnalelor de intrare

$$net = \sum_{i=1}^{n} x_i w_i \qquad (1.1)$$

Funcția de activare f_{AN}, care furnizează semnalul de ieșire y, este în general o funcție monotonă,

$$\{f_{AN}(-\infty)=0 \text{ sau } f_{AN}(-\infty)=-1\}$$

și

$$f_{AN}(+\infty)=1$$

În continuare sunt prezentate câteva dintre cele mai frecvent utilizate funcții de activare.

a) Funcție liniară

$$f_{AN}(net-\theta)=\alpha(net-\theta) \tag{1.2}$$

b) Funcție treaptă

$$f_{AN}(net-\theta)=\begin{cases}\alpha_1, dacă\ net\ \geq\ \theta\\ \alpha_2, dacă\ net\ \leq\ \theta\end{cases} \tag{1.3}$$

Cazurile particulare des întâlnite sunt (α_1=1 şi α_2=0) sau (α_1=1 şi α_2=-1).

c) Funcție rampă

$$f_{AN}(net-\theta)=\begin{cases}\alpha, dacă\ net-\theta\geq\alpha\\ net-\theta, dacă\ |net-\theta|<\alpha\\ -\alpha, dacă\ net-\theta\leq-\alpha\end{cases} \tag{1.4}$$

d) Funcție sigmoidală

$$f_{AN}(net-\theta)=\frac{1}{1+e^{-\lambda(net-\theta)}}\ ,\lambda\geq 0 \tag{1.5}$$

e) Funcție tangentă hiperbolică

$$f_{AN}(net-\theta)=\tanh\big(\lambda(net-\theta)\big)=\frac{e^{\lambda(net-\theta)}-e^{-\lambda(net-\theta)}}{e^{\lambda(net-\theta)}+e^{-\lambda(net-\theta)}} \tag{1.6}$$

f) Funcție ReLU (en. rectified linear unit)

$$f_{AN}(net-\theta)=max(0,net-\theta)=\begin{cases}net-\theta\ , dacă\ net-\theta>0\\ 0\ ,\quad altfel\end{cases} \tag{1.7}$$

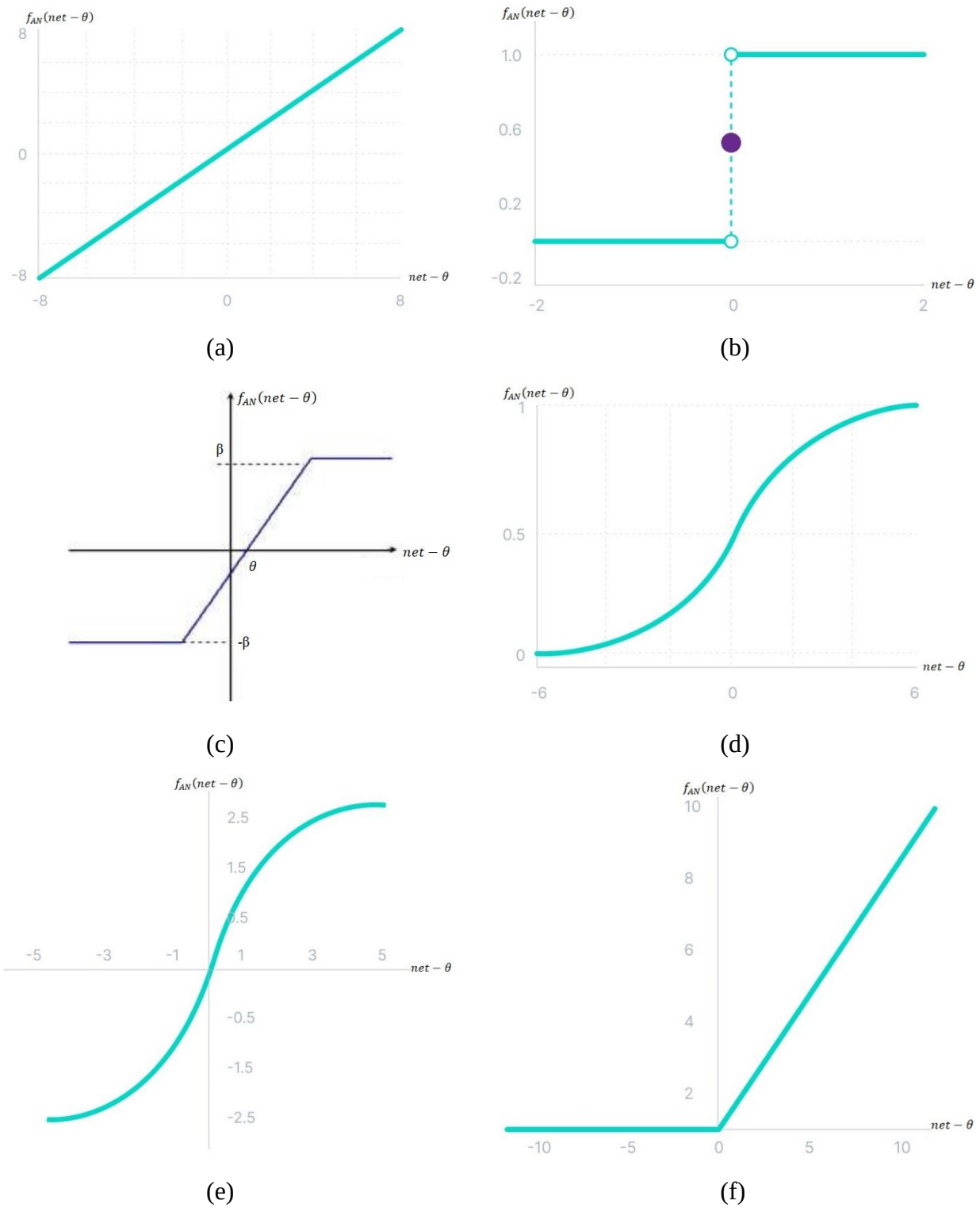


Figura 1.2 Funcțiile de activare ale neuronului ((a) liniară (b) treaptă (c) rampă (d)sigmoidă (e)tangentă hiperbolică (f) ReLU); Sursele: [4], [6]

1.3. Modele de Rețele Neuronale Artificiale

Se numește rețea neuronală artificială (Artificial Neural Network=ANN) cu p straturi un graf orientat etichetat în care vârfurile sunt neuroni v_1, v_2,…v_M dispuși pe p_1 straturi de intrare, p_2 straturi de ieșire și p_3 straturi intermediare, arcele corespund conexiunilor între unii din acești neuroni, pe fiecare arc fiind reprezentată ponderea respectivă. Numărul de neuroni de pe fiecare strat și numărul ponderilor determină arhitectura rețelei neuronale. Rețeaua se numește liniară dacă toți neuronii sunt liniari.

Învățarea reprezintă procesul prin care rețeaua neurală se adaptează la stimuli și reușește să producă răspunsurile dorite în cazul învățării supervizate sau o mulțime consistentă de ieșiri în cazul învățării nesupervizate. Cunoașterea pe care rețeaua o dobândește este memorată în ponderile sinapselor dintre neuroni.

Vectorii destinați antrenării rețelei neurale se prezintă în mod secvențial, iar ponderile sinaptice sunt ajustate pentru a capta cunoașterea pe care acești vectori o conțin.

Ajustarea ponderilor se face cu o procedură prestabilită, o lege sau algoritm de învățare.

Ținând cont de tipul de învățare al rețelelor neuronale artificiale, aceste se clasifică în două tipuri fundamentale:

a. Rețele cu învățare supervizată (pentru antrenare, este utilizat un set de date etichetat)
b. Rețele cu învățare nesupervizată (pentru antrenare, setul utilizat nu este etichetat)

Problemele rezolvate de către rețelele neurale artificiale pot fi de două tipuri:

a. Clasificare – fiind dat un vector de intrare, acesta trebuie încadrat în clasa (categoria) potrivită (ex: clasificarea unor specii de flori pe baza imaginilor acestora)
b. Regresie liniară – fiind dat un vector de intrare, pe baza acestuia trebuie estimată o valoarea continuă (ex: estimarea prețului unui apartament pe baza imaginilor acestuia)

În cazul acestei lucrări, problemele rezolvate de către rețeaua neurală adâncă cu învățare supervizată sunt de regresie liniară. Problemele sunt definite de către setul de date etichetat existent, unde etichetele asociate vectorilor reprezintă codificarea unei valori continue precum magnitudinea cutremurului sau adâncimea hipocentrului.

Eticheta corespunzătoare unui vector reprezintă setul de ieșiri ideale ale rețelei atunci când la intrarea rețelei se aplică vectorul respectiv. Setul de date etichetate de antrenare este echivalent cu un set de exemple de instruire, fiecare exemplu de instruire fiind o pereche formată dintr-un vector de intrare și ieșirea ideală a rețelei. În timpul procesului de învățare, pentru fiecare vector aplicat la intrarea rețelei se calculează ieșirea reală corespunzătoare, iar această ieșire este comparată cu vectorul de ieșire dorit. Se obține un semnal de eroare care este folosit pentru a calcula corecțiile ce trebuie aplicate ponderilor sinaptice astfel încât să se minimizeze această eroare. Tipurile de algoritmi și arhitecturi complet conectate cu învățare supervizată sunt următoarele: perceptron multistrat (Multilayer Perceptron-MLP); rețeaua cu funcții de bază radiale (Radial Basis Function Network-RBF); memoria asociativă bidirecțională (Bidirectional Associative Memory-BAM); rețeaua Hopfield; modelul Deep Learning (învățare adâncă).[7]

1.3.1. Perceptronul Multistrat (Multilayer Perceptron)

Perceptronul este cea mai simplă, cea mai utilizată și totodată printre primele modele de rețele neurale.

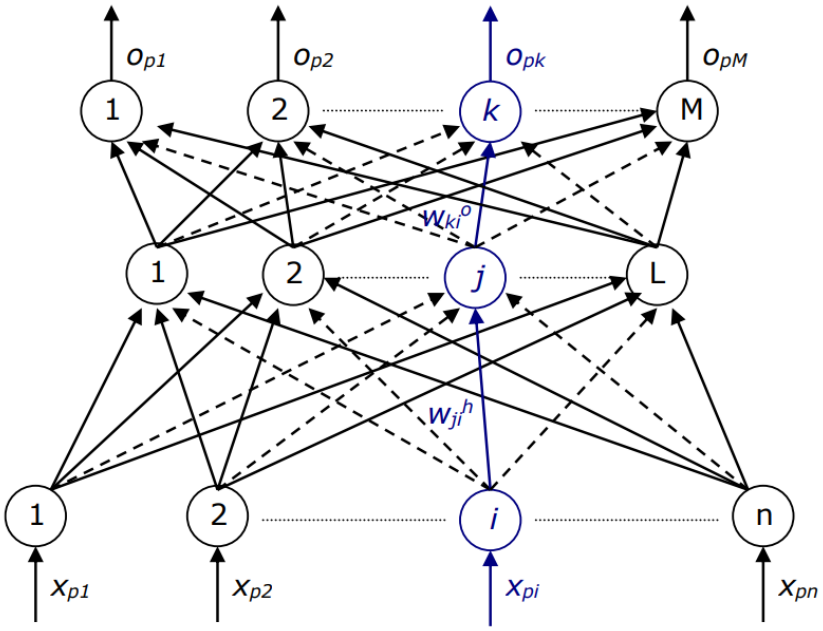


Figura 1.3 Rețea MLP cu trei straturi de neuroni (structură clasică); Sursa: [7]

Structura clasică de MLP este o rețea fără reacție (feed forward), care conține trei straturi de neuroni. Primul strat (input layer) acționează ca un „buffer" al vectorilor de intrare, conținând neuroni virtuali. Al doilea strat, denumit strat ascuns (hidden layer) sau intermediar se comportă ca un detector de caracteristici. Al treilea strat (output layer) conține neuroni clasificatori sau „perceptroni", care combină caracteristicile furnizate de neuronii stratului ascuns pentru a lua o decizie de recunoaștere a formelor.

Forma vectorului de intrare și ecuațiile unui neuron intermediar sau de ieșire sunt la fel precum cele prezentate în cadrul descrierii neuronului artificial[1.2]. Etapele principale în calculul ieșirii unui neuron sunt etapa liniară reprezentată de suma ponderată a intrărilor și etapa neliniară reprezentată de aplicarea funcției de activare rezultatului etapei liniare.

O conexiune între doi neuroni, i→j, se notează astfel:

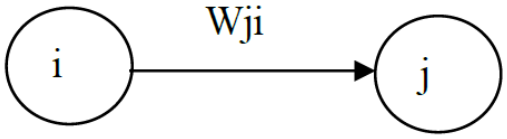


Figura 1.4 Conexiune între doi neuroni; Sursa: [7]

și reprezintă ponderea sinapsei de la neuronul „i" la „j". „j" reprezintă indicele neuronului destinație, iar „i" reprezintă indicele neuronului inițial.

1.3.2. Învățarea supervizată

Învățarea supervizată presupunea rafinarea ponderilor modelului ANN în cursul unui proces de antrenare pe baza unu set de ieșiri ideale aferente fiecărui vector de intrare ce aparține setului special destinat instruirii. Atunci când ieșirea reală a rețelei diferă de cea ideală, se definește eroarea de ieșire (costul) ca diferența dintre cele două ieșiri:

$$E_p = \frac{1}{2}\sum_{k=1}^{M}(y_{pk} - o_{pk})^2 \tag{1.8}$$

unde y_{pk} = ieșirea ideală a neuronului de ieșire „k" când la intrare se introduce vectorul „p" și

o_{pk} = ieșirea reală a neuronului de ieșire „k", în aceleași condiții. Funcția de cost (en. loss function) prezentată anterior este eroarea pătratică medie (en. Mean Square Error – MSE). În practică pot fi folosite multe alte funcții de cost precum eroarea absolută medie (en. Mean Absolute Error – MAE) sau Cross Entropy Loss, pentru problemele de clasificare.

Procesul de învățare al ANN are loc prin aplicarea succesivă a tuturor vectorilor setului de antrenare, în cadrul mai multor epoci, o epocă reprezentând o iterație completă prin întregul set de date de antrenare. Vectorii de antrenare pot fi aplicați la intrarea rețelei neurale fie pe rând, numărul de iterații (pași) al unei epoci fiind egal cu cel al numărului de vectori de intrare, fie, în cazul arhitecturilor de calcul masiv paralele, se poate aplica la intrare loturi de vectori de antrenare (en. batches), urmând ca toți aceștia să fie prelucrați, în paralel de către rețeaua neurală. Astfel, procesare vectorilor de intrare este semnificativ accelerată, scurtând timpul necesar parcurgerii unei epoci de antrenare în cazul seturilor de date de dimensiuni mari.

În cadrul unei epoci, rafinarea ponderilor acționează în sensul modificării (scăderii) erorii globale E_p. Notând cu t indicele iterației, în cazul aplicării seriale al vectorilor la intrarea rețelei neurale, la iterația t = N + p, p < N, la intrarea rețelei se aplică vectorul $\underline{X}(t) = \underline{X}_p$, știind că N reprezintă numărul total de vectori din setul de antrenare. În cazul în care la intrare se aplică loturi de date de o anumită dimensiune, setul de date de antrenare va fi împărțit în mai multe loturi de dimensiunea aleasă și vor fi aplicate, pe rând, la intrarea rețelei neurale în vederea prelucrării datelor acestora, în paralel.

Tehnica de rafinare a ponderilor implică un algoritm de tip „Backpropagation" (ro. propagare inversă), adică procesul de instruire presupune realizarea unei reacții negative, eroarea dintre ieșirea ideală și cea reală fiind propagată în rețea pentru a rafina ponderile acesteia (eroarea se propagă, înapoi, acționând asupra cauzei care a produs-o).

Relația generală de rafinare a ponderilor la momentul „t+1" este:

$$W_{ji}(t+1) = W_{ji}(t) + \Delta W_{ji}(t+1) \tag{1.9}$$

$$\Delta W_{ji}(t+1) = -\eta \frac{\partial E_p}{\partial W_{ji}} \tag{1.10}$$

unde η este rata de învățare, 0 < η < 1. Astfel, prin determinarea dimensiunii și direcției gradientului erorii produse de către ponderea w_{ji}, prin intermediul derivatei parțiale, aceasta poate fi aplicată ca o ajustare în direcția opusă gradientului erorii asupra valorii ponderii de la momentul anterior „t".

Pentru unele aplicații, rata de învățare η poate fi variabilă cu iterația.

Relația generală prezentată anterior este corespunzătoare algoritmului de optimizare Stochastic Gradient Descent (SGD), ce optimizează ponderile la fiecare iterație a unui vector sau a unui lot (en. batch) de vectori de intrare prin rețea.

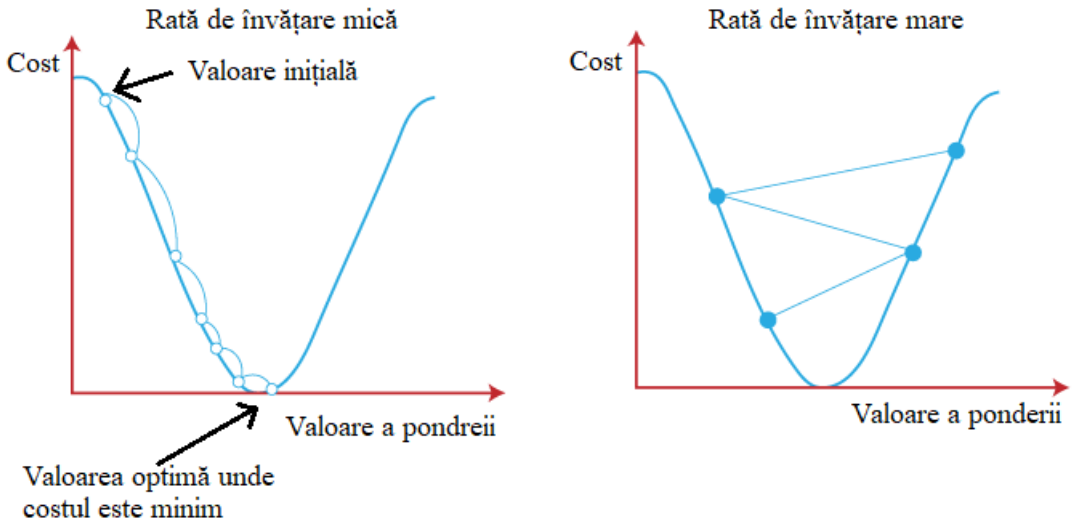


Figura 1.5 Procesul de optimizare al ponderilor folosind SGD

SGD este unul dintre cei mai simpli și cei mai des utilizați algoritmi de optimizare. Alți algoritmi de optimizare frecvent întâlniți în aplicațiile actuale de IC sunt Adam, RMSprop și Adagrad. Mai departe voi continua cu o scurtă descriere a algoritmului de optimizare utilizat în lucrarea actuală, algoritmul Adam (en. Adaptive Moment Estimation).

Algoritmul de optimizare Adam are o abordare diferită față de algoritmul clasic de SGD. SGD menține o singură rată de învățare pentru toate ajustările aduse ponderilor și nu se modifică în timpul antrenării. Adam menține o rată de învățare unică pentru fiecare parametru, și adaptează aceste rate de învățare pe parcursul procesului de instruire.

Algoritmul Adam beneficiază atât de proprietățile algoritmului AdaGrad (en. Adaptive Gradient) cât și de cele ale algoritmului RMSProp (en. Root Mean Squared Propagation).

AdaGrad este un algoritm de optimizare ce menține o rată de învățare pentru fiecare parametru, le ajustează separat în timpul procesului de antrenare și urmărește în acest mod îmbunătățirea performanțelor în cadrul problemelor cu „gradienți dispersi” (en. sparse gradients).[8] Gradienții dispersi se referă la gradienții care au valori sau contribuții mici într-o rețea neurală, ceea ce poate indica o activare slabă sau influență scăzută a unor neuroni în procesul de învățare automată.

RMSProp menține, de asemenea, o rată de învățare unică pentru fiecare parametru, ce se adaptează pe baza mediei magnitudinilor gradienților ponderii respective (ex: în funcție de cât de rapid se modifică într-o anumită direcție).[8] Această adaptare a ponderilor este asigurată de către „momentum” (ro. impuls). Datorită acestor îmbunătățiri, algoritmul RMSProp se descurcă foarte bine în cazul problemelor „online” și al celor „non-staționare” (ex: probleme zgomotoase).

Momentum a fost introdus pentru a reduce varianța algoritmului SGD și pentru a face o tranziție cât mai lină către punctul de optim în spațiul funcției optimizate. Momentum accelerează convergența către direcțiile relevante și reduce fluctuația către direcțiile irelevante. Astfel că în relația generală de optimizare se introduce un nou hyperparametru denumit „momentum” (ro. impuls) și este simbolizat cu γ.

$$W_{ij}(t+1) = \gamma W_{ij}(t) + \Delta W_{ji}(t+1) \tag{1.11}$$

$$\Delta W_{ji}(t+1) = -\eta \frac{\partial E_p}{\partial W_{ji}} \tag{1.12}$$

Momentum are de obicei valoarea 0.9, însă aceasta poate fi ajustă în funcție de necesitățile problemei.

Algoritmul Adam utilizează media momentului de ordin I (media), precum o face și RMSProp, însă acesta profită în ajustarea ponderilor și de media momentelor de ordin II (varianța necentrată) a gradienților. Algoritmul calculează o „medie glisantă exponențială" (en. exponential moving average) a gradientului și a gradientului pătrat, iar parametrii beta1 și beta2 controlează ratele de degradare ale celor două medii glisante.[8]

Ideea din spatele algoritmului Adam este aceea că „impulsul" în procesul de actualizare al ponderilor nu trebuie să fie foarte mare, acest lucru putând duce la un salt peste minimul dorit. Viteza de actualizare a ponderilor în direcția dorită trebuie redusă pentru a produce o căutare exhaustivă.[9]

Algoritmul Adam deși este unul cu un cost computațional ridicat, viteza mare de convergență, stabilitatea pașilor de optimizare prin reducerea variației și rectificarea reducerii ratei de învățare fac ca algoritmul să fie una din soluțiile optime pentru foarte multe aplicații de IC.

1.4. Modelul Deep Learning

Învățarea profundă (en. Deep Learning) este un subset al Inteligenței Computaționale, specific focalizat pe utilizarea rețelelor neuronale cu capacitatea de extragere în mod automat a trăsăturilor utile și a formelor (en. patterns) incluse în interiorul datelor. Trăsăturile extrase din datele neprelucrate sunt utilizate pentru a compune o sarcină de învățare informată și a lua decizii viitoare pe baza cunoștințelor acumulate. Pentru a informa deciziile, mai întâi este necesară învățarea modului de extragere al trăsăturilor din datele de intrare, trăsături care vor determina cum trebuie îndeplinită cu succes sarcina algoritmului.

Algoritmii tradiționali de învățare automată (en. Machine Learning), de obicei operează prin definirea unui set de reguli sau caracteristici în mediul datelor. Aceste reguli sunt de obicei definite manual de către un operator uman care va analiza datele și va încerca să extragă trăsăturile principale ale datelor. Acest proces de explorare al datelor în vederea extragerii caracteristicilor va avea o influență semnificativă asupra performanțelor algoritmilor de învățare, ei depinzând în mod sensibil de reprezentarea datelor.

Algoritmii de Deep Learning urmăresc rezolvarea acestor probleme cauzate de extragerea în mod eficient a caracteristicilor relevante din setul de date, abordarea fiind una cu totul diferită. Cheia rezolvării cu succes a sarcinilor este extragerea și învățarea trăsăturilor direct din setul de date, într-un mod ierarhic. Această extragere a trăsăturilor într-un mod ierarhic presupune ca fiind dat un set de date, spre exemplu un set de date cu sarcina de identificare a fețelor, se pune problema implementării unui model de rețea neurală care să primească ca dată de intrare o imagine, iar pe baza acesteia să detecteze diferite trăsături definitorii ale unei fețe. Primul pas este detectarea elementelor de bază, a contururilor (caracteristici de nivel jos), apoi construirea pe baza contururilor a nasului, ochilor și gurii (caracteristici de nivel mediu), iar cu ajutorul caracteristicilor de nivel mediu se vor compune elementele definitorii ale structurii faciale (caracteristici de nivel înalt). Astfel, pe măsură ce informația este procesată de straturile unei rețele neurale adânci, se va distinge abilitatea de a capta astfel de trăsături ierarhice, reprezentând avantajul învățării profunde comparativ cu învățarea automată clasică. Rețelele neurale adânci realizează extragerea caracteristicilor și reprezentarea lor

dintr-un spațiu inițial complex, neprelucrat, într-un nou spațiu mai simplu, ce conține doar trăsăturile esențiale ale datelor de intrare.

Blocurile constructive fundamentale ale învățării profunde și algoritmii de bază există de decenii, însă domeniul a început să ia amploare de curând deoarece datele au început să fie din ce în ce mai răspândite. Datele sunt puterea motrice a foarte multor astfel de algoritmi iar azi ne uităm la o lume în care avem mai multe date ca oricând. Facilitatea colectării și stocării seturilor de date cât mai mari și mai diverse, de complexități ridicate, a condus la nașterea domeniului Big Data.

Un alt motiv pentru care domeniul a început de curând să capete atenție este că acești algoritmi și aceste modele de rețele neurale sunt masiv paralelizabile. Până acum puterea de calcul și arhitectura sistemelor de calcul (serială) nu era suficientă pentru a putea implementa aceste modele și acești algoritmi foarte adânci. Datorită avansurilor ce au avut loc în ultimele decenii în domeniul arhitecturilor multi-CPU și multi-GPU, dar mai ales în domeniul procesoarelor grafice, acum modelele neurale pot beneficia de avantajele majore aduse de noile tehnologii precum framework-ul NVIDIA Compute Unified Device Arhitecture (CUDA). CUDA este o platformă de calcul paralelă ce permite dezvoltatorilor să accelereze dramatic aplicațiile de calcul prin valorificare puterii arhitecturii paralele a procesoarelor grafice.

În ultimul rând, cu ajutorul uneltelor software open source precum PyTorch sau Tensorflow este posibilă construirea și aplicarea algoritmilor complecși de DL mult mai facil, iar prin utilizarea acestor biblioteci software alături de framework-ul CUDA timpul de antrenare și testare al rețelelor neuronale adânci este redus semnificativ.

1.4.1. Rețele neurale convoluționale

Rețelele neurale convoluționale (en. Convolutional Neural Networks – CNN) sunt modele neurale multistrat specializate în rezolvarea de probleme de detecție a formelor (en. pattern recognition), fiind dezvoltări inspirate din biologie ale modelului Perceptron Multistrat. Comparat cu rețelele prezentate anterior de tip feedforward cu un același număr de straturi, modelul CNN are mult mai puține conexiuni și, implicit, un număr de parametri mult mai redus. Acest lucru conduce la o antrenare mult mai simplă și totodată la performanțe superioare de recunoaștere.

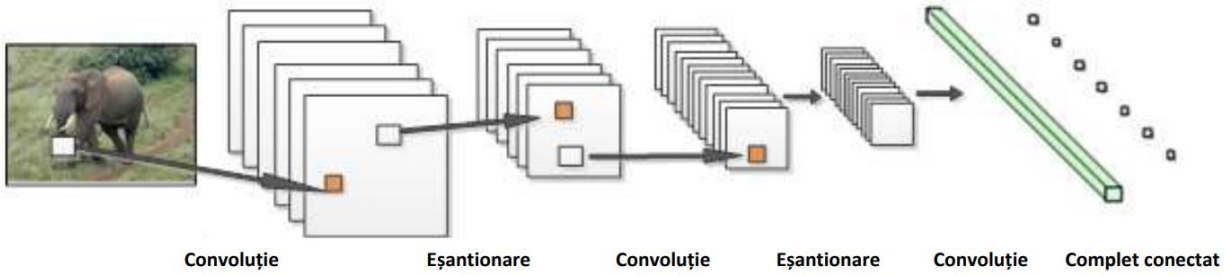


Figura 1.6 Structură CNN; Sursa: [10]

CNN-urile sunt rețele neurale ierarhice ce se bazează pe operații de convoluție intercalate cu straturi de interpolare (eșantionare), având drept scop extragerea caracteristicilor esențiale precum și reducerea dimensiunii datelor. Operația finală de clasificare (sau regresie liniară) este realizată de un număr de straturi complet conectate (fully connected) introduse la finalul rețelei convoluționale, după extragerea trăsăturilor esențiale de nivel înalt.

1.4.1.1. Strat de convoluție

Stratul de convoluție este descris de numărul de hărți de caracteristici (en. feature maps) dat de numărul de nuclee (filtre) utilizate în acel strat, dimensiunea nucleelor (kernels) și conexiunile cu stratul precedent. Adițional se ține cont de modul în care are loc convoluția, dacă matricea este bordată și care este dimensiunea bordării (en. padding), alături de dimensiunea saltului ferestrei de convoluție (en. stride).

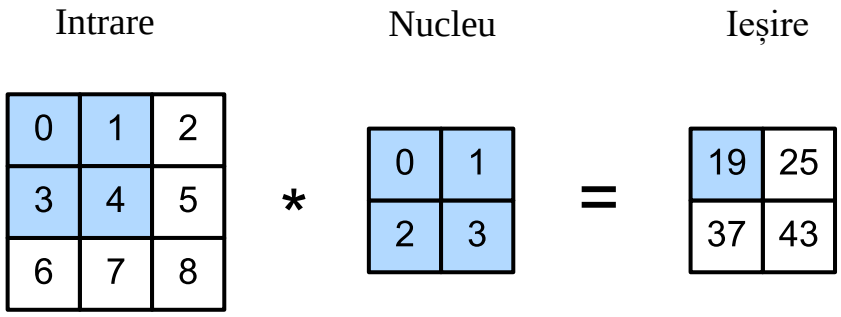


Figura 1.7 Operația de convoluție; Sursa: [11]

Fiecare strat conține un număr M de hărți de caracteristici cu aceeași dimensiune (M_x, M_y), fiecare hartă de caracteristici este un rezultat al sumei convoluțiilor hărților de caracteristici din stratul precedent cu nucleele lor corespondente și procesate de un filtru liniar. La final se adaugă un termen de bias (ro. pierdere) și se aplică o funcție de activare neliniară

$$M_{i,j}^{k} = ReLU\left((W^{k} * x)_{i,j} + b_{k}\right), \tag{1.13}$$

unde M^k este harta caracteristicilor cu index k, caracterizată prin ponderile W^k și termenul de bias b_k, folosind funcția ReLU.

Dimensiunile hărții de ieșire sunt:

$$M_{x}^{k} = \frac{M_{x}^{k-1} + 2P_{x}^{k} - K_{x}^{k}}{S_{x}^{k}} + 1 \tag{1.14}$$

$$M_{y}^{k} = \frac{M_{y}^{k-1} + 2P_{y}^{k} - K_{y}^{k}}{S_{y}^{k}} + 1 \tag{1.15}$$

unde indexul k indică stratul, K_x și K_y sunt dimensiunile nucleului, P_x și P_y sunt dimensiunea bordării, iar S_x și S_y sunt numărul de pixeli săriți de nucleu în procesul de convoluție denumiți stride sau skipping factors.

Convoluția unui semnal bidimensional discret este definită prin următoarea relație:

$$o[m,n] = f[m,n] * g[m,n] = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f[u,v]g[u-m, v-n] \tag{1.16}$$

Neuronii dintr-o hartă de caracteristici își împart ponderile prin reutilizarea nucleelor în straturile convoluționale, acest fapt are ca efect creșterea eficienței prin reducerea numărului de parametrii antrenabili. Ca rezultat, CNN-urile vor avea un nivel de generalizare mai ridicat.

1.4.1.2. Strat de Pooling

Stratul de Pooling este un strat de eșantionare ce precedă stratul convoluțional. Cel mai frecvent utilizat tip de strat de Pooling este cel de tip Max-Pooling, ce reușește să scoată în evidență și să păstreze doar trăsăturile cele mai importante extrase în urma convoluției (ex: contururile).

Acest strat reduce dimensiunea hărților, îmbunătățind viteza de convergență (reduce efortul computațional) și crește capacitatea de generalizare datorită invarianței la poziție.

Ieșirea unui strat de Max-Pooling este dată de valoarea maximă de activare din regiunile non-overlapping ale nucleelor, venind ca o alternativă în locul abordării clasice de mediere a intrărilor, Average-Pooling. Un strat de Average-Pooling are ca ieșire media valorilor din interiorul regiunilor non-overlapping ale nucleelor.

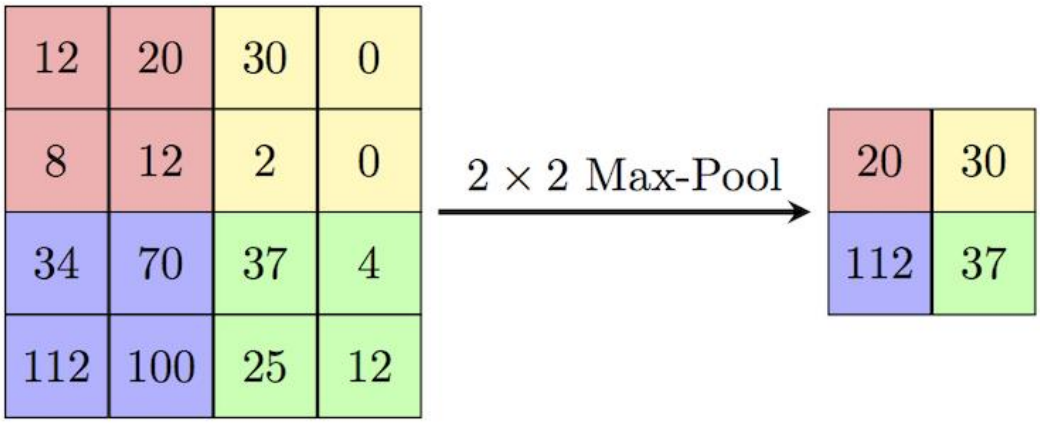


Figura 1.8 Operația de Max-Pooling cu fereastră 2x2 și 2 pixeli săriți; Sursa: [12]

1.4.1.3. Strat de clasificare/regresie

Se aleg parametrii modelului pentru straturile de convoluție și max-pooling astfel încât ieșirile ultimului strat convoluțional să formeze un vector unidimensional de caracteristici. O altă alternativă ar fi transformarea vectorului de trăsături rezultat la ieșirea ultimului strat convoluțional într-un vector unidimensional (en. flatten) și alegerea numărului de neuroni al primului strat complet conectat ca fiind egal cu dimensiunea vectorului unidimensional.

Acest vector de caracteristici va fi folosit ca intrare pentru o serie de straturi complet conectate (en. fully connected) care vor executa o operație de clasificare sau regresie liniară, în funcție de problema dată.

Ultimul strat din acest grup are câte un singur neuron pentru fiecare clasă sau câte un neuron pentru fiecare valoare continuă ce trebuie estimată prin regresie liniară.

Capitolul 2. Metoda abordată

2.1. Reprezentarea semnalelor seismice în domeniul timp-frecvență

Majoritatea soluțiilor de ultimă oră în tratarea problemelor de vedere artificială implică utilizarea rețelelor neurale convoluționale. Sarcinile legate de analiza și recunoașterea imaginilor implică suprapunerea de straturi convoluționale ce permit extragerea trăsăturilor semnificative, precum a fost prezentat în 1.4.1.

În domeniul analizei semnalelor seismice se poate urma o abordare similară cu cea a domeniului vederii artificiale. Pentru analiza semnalelor seismice se va transforma semnalul din domeniul timp într-un semnal bidimensional în domeniul timp-frecvență prin aplicarea Transformatei Fourier de Timp Scurt (Short-time Fourier Transform - STFT) și se va converti spectrograma obținută din valori de amplitudine într-o reprezentare logaritmică (dB). Pe baza spectrogramei obținute se vor extrage trăsăturile semnalelor seismice și se va realiza analiza semnalului seismic.

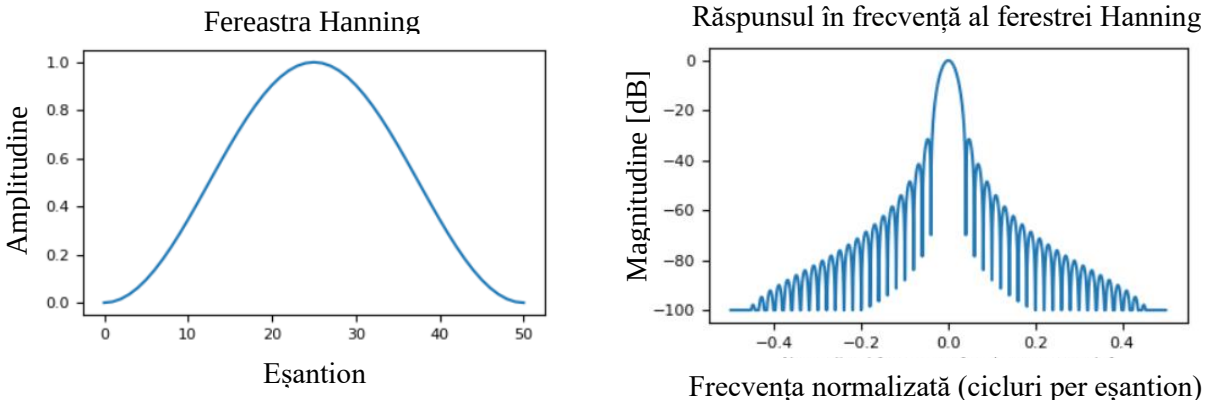
Transformata Fourier de Timp Scurt este o transformată Fourier folosită pentru a determina frecvența sinusoidală și conținutul de fază al secțiunilor locale ale unui semnal pe măsură ce variază în timp. În practică, procedura de calcul STFT este de a împărți semnalul în domeniul timp în segmente de lungime egală și apoi de a calcula Transformata Fourier de Timp Discret (TFDT), în cazul semnalelor digitale, pe fiecare segment, separat. Transformata dezvăluie spectrul Fourier al fiecărui segment.[13] Transformata Fourier de Timp Scurt pentru un semnal discret este următoarea:

$$STFT\{x[n]\}(k,r) \equiv X(k,r) = \sum_{n=0}^{L-1} x[rR+m] \cdot w[m] e^{-j\frac{2\pi}{N_x}km} \qquad (2.1)$$

unde x[n] este semnalul discret, w[n] este funcția fereastră de lungime L, N_x este lungimea STFT, iar R este dimensiunea pasului de calcul.[2] În majoritatea aplicațiilor practice, un sistem de calcul utilizează Transformata Fourier Rapidă (en. Fast Fourier Transform) care utilizează valori discrete și cuantizate atât pentru dimensiunea ferestrei, cât și pentru frecvență.

Pentru simplitate, voi utiliza tipul de fereastră oferit standard de către biblioteca open-source librosa pentru funcția „librosa.stft". Fereastra implicită a acestei funcții este fereastra cosinus ridicat (sau Hanning), o fereastră indicată pentru aplicațiile de procesare a semnalelor audio.

Pentru a realiza analiza în domeniul timp-frecvență a semnalelor seismice, alte reprezentări ale domeniului timp frecvență ar putea fi considerate, de exemplu, transformata continuă wavelet cu diferite funcții de bază sau transformata S. Cu toate acestea, considerând complexitatea computațională mult mai scăzută a STFT comparativ cu celelalte metode, mă voi rezuma la a utiliza această transformare în reprezentarea semnalelor în domeniul timp-frecvență.



$$w(n) = 0.5 - 0.5\cos\left(\frac{2\pi n}{M-1}\right) \qquad 0 \le n \le M-1$$

Figura 2.1 Fereastra Hanning

De obicei, semnalele seismice sunt înregistrate pe trei direcții: Est-Vest (E), Nord-Sud (N) și perpendicular (Z). Prin urmare, semnalele seismice sunt vectori s[n] cu trei componente de dimensiunile (3, N), unde n este variabila de timp discret, iar N este lungimea semnalului. În consecință, STFT va fi calculată pentru fiecare direcție, separat, iar rezultatele vor fi concatenate într-un Tensor multidimensional.

Tensorii PyTorch sunt tipuri de date similare cu array-urile Numpy, singura diferență majoră dintre aceștia fiind aceea că un Tensor PyTorch poate fi executat atât pe CPU, cât și pe GPU.

Reprezentarea vizuală a variației în domeniul timp al spectrului de frecvențe al unui semnal seismic se realizează cu ajutorul unei spectrograme. Astfel că, uneori, rezultat Transformatei Fourier de Timp Scurt este denumit spectrogramă, deși o spectrogramă este doar reprezentarea vizuală a acestui rezultat.

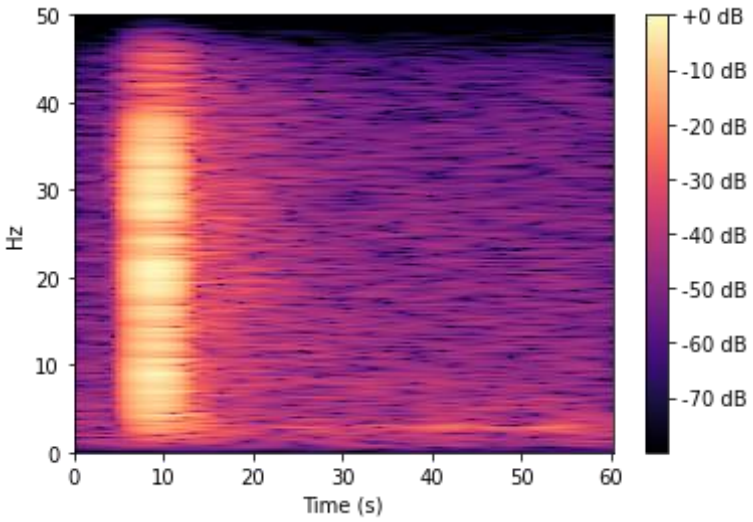


Figura 2.2 Spectrograma seismului pe orientarea Est-Vest

Tensorul multidimensional ce conține spectrogramele semnalului seismic va servi drept Tensor de intrare în rețeaua neurală adâncă de tip convoluțional implementată. Acesta va permite analiza și estimarea parametrilor sursei de semnal seismic, conform sarcinilor propuse în Introducere.

2.2. Arhitectura rețelei neurale

Modelul de CNN ce va fi utilizat în analiza semnalelor seismice este inspirat din abordarea propusă de către Ristea N. și Rădoi A. în articolul „Complex Neural Networks for Estimating Epicentral Distance, Depth, and Magnitude of Seismic Waves”. Soluția adusă în cadrul articolului este implementarea unei arhitecturi neural convoluționale de tip ResNet ce permite utilizarea unui număr ridicat de straturi convoluționale.

Transformările timp-frecvență precum STFT sau transformata S sunt aplicate pentru a obține reprezentări semnificative ale datelor seismice. Cu toate acestea, rezultatele unor astfel de transformări sunt valori în domeniul complex. Prin urmare, în cadrul articolului este urmată o cale a implementării unei rețele neurale adânci ce operează cu date complexe. Prin prelucrarea paralelă a trăsăturilor reale și imaginare ale semnalelor de către arhitectura complexă a rețelei neurale vor putea fi extrase mult mai multe informații relevante din spectrul de frecvențe al semnalului seismic.

Abordarea mea va omite partea imaginară rezultată în urma calculului STFT și va folosi doar de rezultatul real al acestei operații. Ipoteza lucrării de față este major simplificată comparativ cu cea abordată în cadrul articolului. Analiza unei singure zone geografice specifice face suficientă utilizarea părții reale a STFT în obținerea unor rezultate comparabile cu cele obținute pe tot setul de date global.

O rețea reziduală (en. Residual Network – ResNet) este un model de arhitectură DL utilizat cel mai des în cadrul aplicațiilor de vedere artificială (en. Computer Vision). Este o arhitectură CNN proiectată să suporte sute sau chiar mii de straturi convoluționale. Arhitecturile precedente de CNN nu aveau capacitatea de a se adapta la numărul mare de straturi, lucru care conducea la o limitare a performanțelor pe măsura creșterii dimensiunii rețelei neurale. Prin adăugarea unui număr din ce în ce mai mare de straturi, cercetătorii s-au lovit de problema „risipirii gradienților” (en. vanishing gradients). [14]

Precum a fost prezentat în capitolul 1.3.2, rețelele neurale sunt antrenate printr-un algoritm de backpropagation ce se bazează pe algoritmul de optimizare gradient descendent (en. gradient descent). Pe baza erorii găsite cu ajutorul funcției de cost, se procedează în ajustarea ponderilor în vederea minimizării erorii (în cazul lucrării de față, funcția de cost utilizată va fi MSE). Dacă numărul de straturi al rețelei este mult prea mare, înmulțirile multiple în procesul de propagare al erorii și de optimizare al ponderilor vor reduce eventual în mod considerabil gradienții, aceștia „risipindu-se”. În final, straturile aflate la începutul rețelei neurale vor primi ajustări mult prea mici, performanța totală a rețelei saturându-se sau chiar deteriorându-se cu fiecare strat adăugat suplimentar.

ResNet vine cu o soluție inovativă în vederea rezolvării problemei risipirii gradienților, cunoscută sub numele de „salturi de conexiune” (en. skip connections). ResNet suprapune mai multe mapări de identitate (salturi de conexiune), ce omit anumite straturi convoluționale și reutilizează rezultatele stratului anterior, obținute în urma aplicării funcției de activare. Salturile accelerează procesul de antrenare inițial prin compresia rețelei neurale. Apoi, când rețeaua este reantrenată, toate straturile sunt utilizate, iar părțile rămase neantrenate din rețea, cunoscute sub numele de straturi reziduale, vor permite explorarea mult mai detaliată a spațiului caracteristicilor setului de date de intrare.

Blocurile Reziduale reprezintă inovația adusă de către arhitectura rețelelor ResNet. În arhitecturile anterioare precum VGG16 [15] sau LeNet [16] straturile convoluționale sunt suprapuse cu straturi de normalizare a lotului (en. batch normalization) și funcții de activare neliniare. Această metodă funcționează doar pentru arhitecturile cu număr mic de straturi convoluționale.

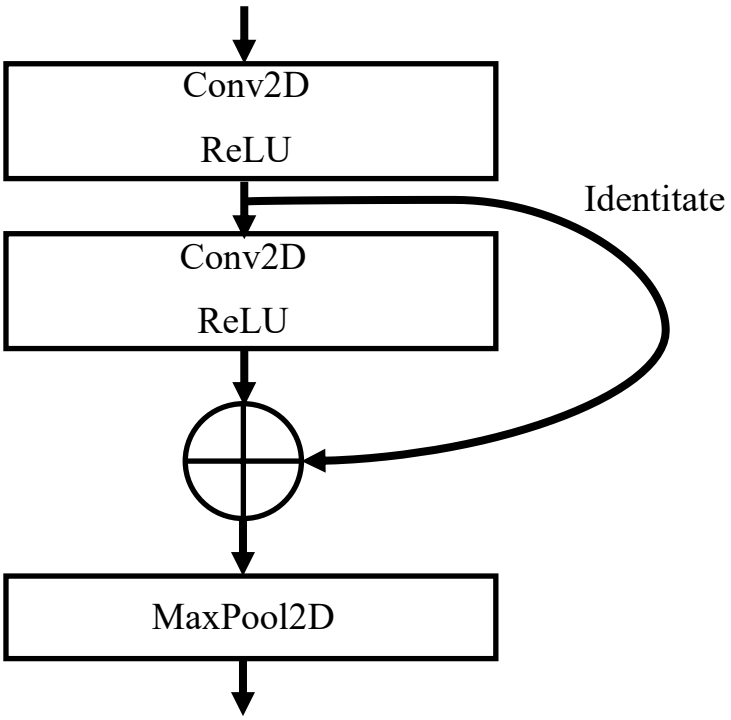


Figura 2.3 Bloc Rezidual al ResNet cu salt peste un strat

Arhitectura ResNet introduce un concept simplu, adăugarea de intrări intermediare la ieșirea unei serii de blocuri convoluționale. În cazul de față, în cadrul unui bloc rezidual, un strat convoluțional este tratat precum în arhitecturile menționate anterior, iar cel de-al doilea este un strat convoluțional rezidual, antrenat mult mai târziu un procesul de antrenare. Astfel rețeaua permite extragerea de trăsături și relații mult mai complexe în interiorul semnalelor seismice de intrare.

Exact cum a fost menționat și anterior, rețelele neurale adânci precum VGG16 sau ResNet au straturile convoluționale suprapuse cu straturi de normalizare a loturilor și funcții de activare neliniare. Normalizarea loturilor este o tehnică de regularizare utilizată pentru a reduce varianța din interiorul datelor și a crește generalizarea soluției adusă de către rețea în tratarea problemei.

Tehnicile de regularizarea au rolul de a îmbunătății performanțele modelului și permit acestuia să conveargă mai rapid către soluția optimă. Câteva dintre instrumentele de regularizare sunt „early stopping”, „dropout”, tehnicile de inițializare a ponderilor și normalizarea loturilor. Regularizarea previne supraînvățarea modelului și face ca procesul de instruire să fie mai eficient. Despre supraînvățare se va discuta în Capitolul 4. Experimente.

Normalizarea este un instrument de preprocesare a datelor folosit pentru a aduce datele numerice la o scară comună, fără a le distorsiona forma. În general, atunci când sunt introduse date într-un algoritm de învățare profundă, apare tendința de a schimba valorile la o scară echilibrată. Motivul pentru care datele sunt normalizate este pentru a asigura că modelul va putea generaliza în mod corespunzător.[17]

Revenind la normalizarea loturilor, acesta este un proces ce face rețelele neurale mai rapide și mai stabile prin adăugarea de straturi suplimentare într-o rețea neuronală adâncă. Noul strat, „BatchNorm”, realizează operațiile de standardizare și normalizare a datelor de intrare provenite dintr-un strat anterior. Această operație are loc asupra unui întreg lot de date de intrare.

Un alt avantaj major adus de către normalizarea loturilor, pe lângă cele menționate anterior, este rezolvarea problemei „deplasării interne a covariatelor” (en. internal covariate shift). Prin această modalitate este asigurat că intrarea fiecărui strat este distribuită uniform în jurul aceleiași medii și

deviații standard, rețeaua producând o generalizare mult mai bună.[17] Capacitatea ridicată de învățare și generalizare a rețelei neurale de tip ResNet datorată arhitecturii specifice descrise anterior, produc o extragere extrem de eficientă a trăsăturilor din datele de intrare neprelucrate.

Astfel, o rețea reziduală reprezintă soluția optimă pentru analiza și prelucrarea semnalelor seismice. Forma finală a implementării rețelei neurale adânci propuse de mine este următoarea:

Bloc	Straturi	Dimensiune ieșire
Conv & Pool 1	Conv2D (7 x 7, 16) ReLU MaxPool2D(2x2)	16 x 256 x 94
Conv & Pool 2	Conv2D (7 x 7, 16) ReLU MaxPool2D(2x2)	16 x 128 x 47
Rezidual 3	Conv2D (5 x 5, 32) ReLU Conv2D (5 x 5, 32) ReLU MaxPool2D(2x2)	32 x 64 x 23
Rezidual 4	Conv2D (3 x 3, 64) ReLU Conv2D (3 x 3, 64) ReLU MaxPool2D(2x2)	64 x 32 x 11
Rezidual 5	Conv2D (3 x 3, 96) ReLU Conv2D (3 x 3, 96) ReLU MaxPool2D(2x2)	96 x 16 x 5
Conv & Pool 6	Conv2D (3 x 3, 128) ReLU Conv2D (3 x 3, 128) MaxPool2D(2x2)	128 x 8 x 2
FC 1	Complet Conectat	1 x 1024
FC 2	Complet Conectat	1 x 4

Tabel 2.1 Structura detaliată a arhitecturii CNN

Implementarea rețelei neurale adânci propusă de mine utilizează 6 blocuri convoluționale, dintre care 3 blocuri reziduale de maniera prezentată în Figura 2.3. Acestea utilizează o identitate ce omite unul din cele două straturi convoluționale aferente blocului rezidual, rezolvând astfel eventualele dificultăți puse de către problema risipirii gradienților.

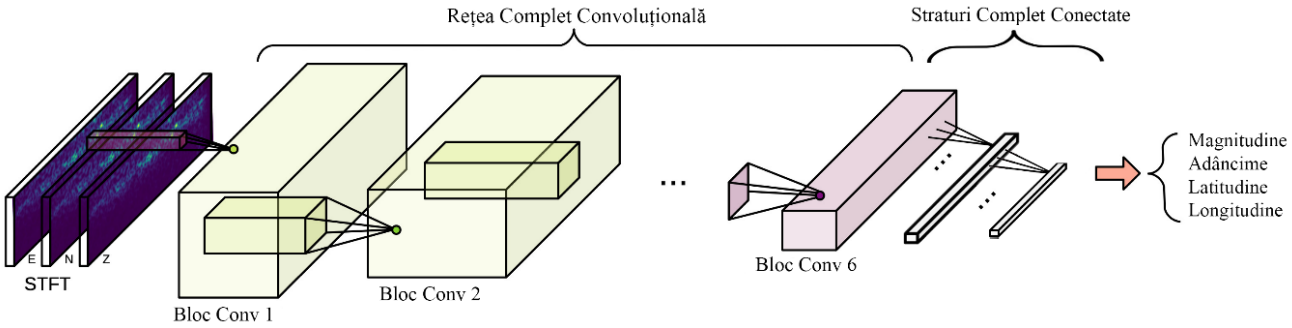


Figura 2.4 Schema generală a rețelei convoluționale propuse

În final, pentru îndeplinirea sarcinii de regresie liniară și estimare a parametrilor sursei seismice sunt utilizate două straturi complet conectate, de forma celor prezentate în cadrul secțiunii 1.3.1. Primul strat complet conectat are ca funcție de activare ReLU, iar cel de-al doilea strat complet conectat are funcția de activare liniară. Numărul de neuroni de ieșire ai ultimului strat este egal cu numărul de parametrii estimați de către rețea. În cazul de față este vorba despre cei patru parametrii stabiliți în introducerea lucrării: magnitudine, latitudine, longitudine și adâncime.

Arhitectura va fi implementată atât sub forma unui LeNet, fără salturile de conexiune prezentate și fără normalizarea loturilor, cât și în arhitectura ResNet. Astfel, se vor putea studia îmbunătățirile în performanță aduse de blocurile reziduale față de arhitectura simplă a LeNet și dacă aceste îmbunătățiri sunt suficient de semnificative în comparație cu resursele necesare antrenării și utilizării în practică a arhitecturii ResNet.

Capitolul 3. Setul de date experimental

3.1. Descrierea seismelor și a undelor seismice

Undele seismice sunt generate de către cutremure și sunt înregistrate sub formă de seismograme de către aparate speciale de măsură denumite aparte seismograf. Seismogramele sunt înregistrări ale mișcării solului într-un loc particular ca funcție de timp. Pentru a caracteriza vectorul de componente al mișcării solului, cutremurele sunt adesea înregistrate de către instrumente cu trei componente, echipate cu un senzor vertical și alți doi senzori orizontal ortogonali (Figura 3.1). [3]

Mai multe sosiri de unde seismice, denumite faze, pot fi observate pe seismograme. Fazele P și S sunt cele două momente seismice fundamentale observabile pe seismograma unui cutremur. În faza P sau a undelor de compresiune (longitudinale), materialul se mișcă înainte-înapoi pe direcția de propagare a undei seismice, pe când în faza S sau a undelor de forfecare (transversale), materialul se mișcă în unghiuri perpendiculare cu direcția de propagare. Undele P se propagă mai rapid decât undele S, astfel că primul puls sosit, etichetat „P”, este o undă P ce a urmat o cale directă de la cutremur la stația seismică.[3] Cu toate acestea, undele fazei S sunt cele mai distructive, producând cele mai multe pagube materiale și făcând cele mai multe victime în cazul cutremurelor de mare magnitudine.

Un cutremur începe să se producă la hipocentru (sau focar). Hipocentrul este definit de o poziție la suprafața pământului (epicentru) și o adâncime dedesubtul acestui punct. Hipocentrul unui cutremur este aflat din timpii de sosire ai undelor seismice înregistrați pe seismometre aflate în locuri diferite.

Mărimea unui cutremur la sursa care l-a produs este măsurată din amplitudinea (sau uneori durata) mișcării înregistrate pe seismograme și este exprimată în magnitudine. Magnitudinea este o mărime logaritmică. La aceeași distanță față de un cutremur, amplitudinea undelor seismice din care este determinată magnitudinea sunt de 10 ori mai mari în timpul unui cutremur de magnitudine 5 față de un cutremur de magnitudine 4. Cantitatea totală de energie eliberată de un cutremur mediu, în funcție de tipul de magnitudine, crește cu un factor de aproximativ 32 pentru fiecare creștere unitară a magnitudinii.

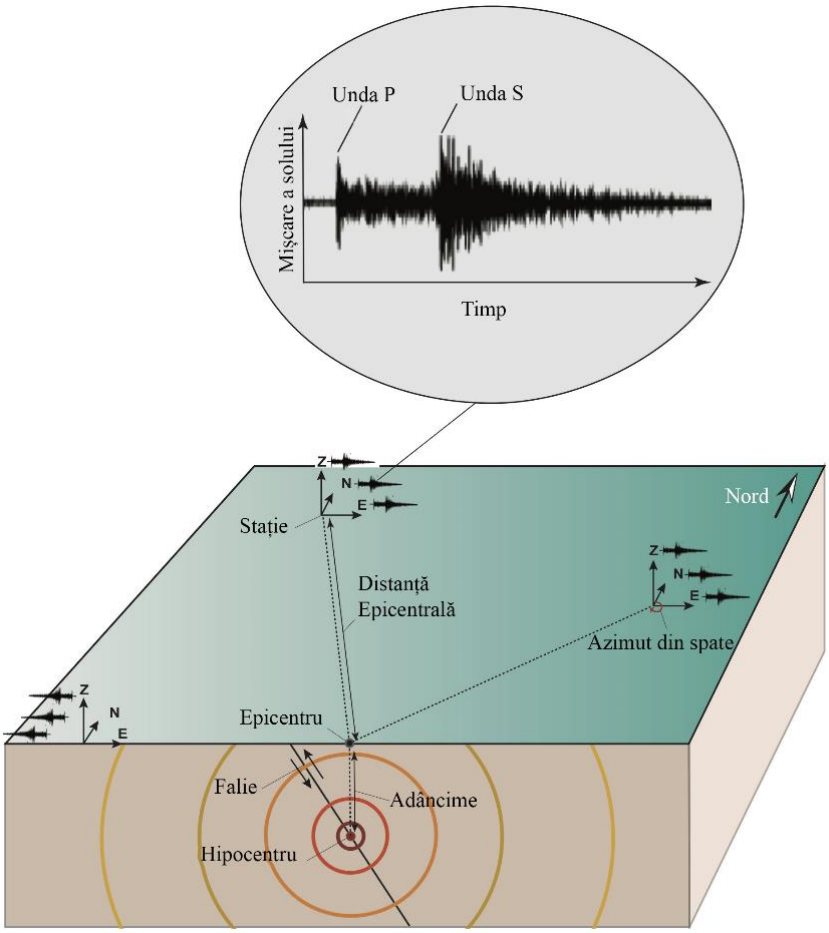


Figura 3.1 Schema propagarea a undelor seismice și înregistrarea mișcării solului de către stații; Sursa: [3]

3.2. Descrierea setului de date STEAD

Experimentele din cadrul acestei lucrări sunt desfășurate folosind un set de date public pus la dispoziție de către Universitatea din Stanford, STanford EArthquake Data set (STEAD).

Setul de date STEAD este unul extrem de vast și diversificat. Acesta conține două clase principale de semnale înregistrate de către instrumentele seismice: cutremure și non-cutremure. Clasa de cutremure conține 1,05 milioane de seismograme formate din trei componente (cu durata de un minut) asociate cu aproximativ 450,000 de cutremure ce au avut loc în perioada Ianuarie 1984 – August 2018. Cutremurele din setul de date sunt înregistrate de către 2613 seismometre localizate global la distanțe de 350 km față de sursele seismice (cutremure local).[3]

Datele seismice sunt furnizate drept NumPy arrays individuale ce conțin trei forme de undă: fiecare formă de undă este formată din 6000 de eșantioane asociate cu 60 de secunde de mișcare a solului înregistrate pe direcțiile Est-Vest (E), Nord-Sud (N) și Verticală (Z). Fiecare dată seismică de tip NumPy array vine însoțită de 35 de etichete, în cazul cutremurelor. Dintre cele 35 de etichete, doar câteva vor fi relevante în cadrul procesului de antrenare și testare al rețelelor neurale adânci.

În figura următoare, va fi prezentat un exemplu de seismogramă al unui cutremur alături de cele 35 de etichete ale sale. Etichetele esențiale în procesul construire al setului de date personalizat și în procesul de antrenare al rețelei neurale adânci sunt următoarele: codul receptorului

(„reciever_code”), latitudinea receptorului („reciever_latitude”), longitudinea receptorului („reciever_longitude”), altitudinea receptorului („reciever_elevation_m”), tipul receptorului („reciever_type”), magnitudinea sursei („source_magnitude”), latitudinea sursei („source_latitude”), longitudinea sursei („source_longitude”) și adâncimea sursei („source_depth”). Receptorul reprezintă instrumentul de măsură utilizat, iar sursa este seismul ce a fost înregistrat.

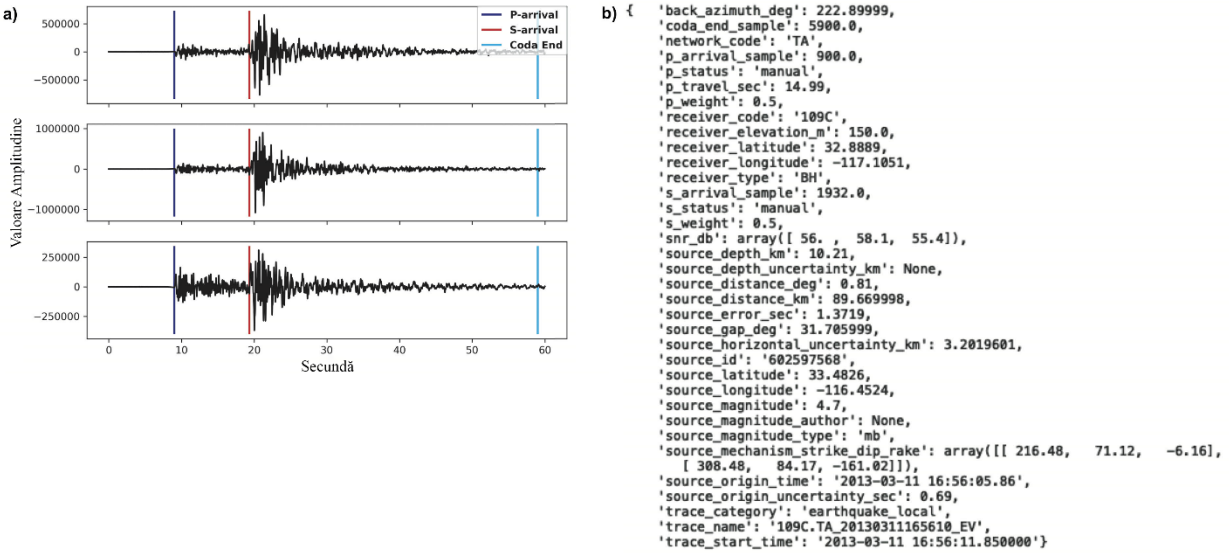


Figura 3.2 Exemplu de seismogramă a unui cutremur. a) mișcarea solului pe direcțiile Est-Vest, Nord-Sud și Verticală în domeniul timp. b) etichetele și informațiile asociate; Sursa: [3]

Atât cutremurele cât și non-cutremurele (zgomotele) vin însoțite de informațiile legate de instrumentele de măsură (ex. codul de rețea, codul instrumentului, tipul și locația seismometrului). Adițional etichetelor anterioare, datele ce reprezintă cutremure conțin în rândul etichetelor informații legate de cutremur (ex. ora de origine, locația epicentrului, adâncimea, magnitudinea, tipul magnitudinii, mecanism focal, timpul de sosire al fazelor P și S, erori estimate, etc.) și semnalul înregistrat (ex. măsurători ale RSZ ale fiecărei componente, sfârșitul energiei dominante a semnalului (coda-end) și distanța față de epicentru).

Unitatea de măsură a fiecărui atribut (etichete) este inclusă în numele acestuia. Epicentrele cutremurelor („source_latitude” și „source_longitude”) sunt date în unități de latitudine și longitudine în cadrul de referință WGS84. Adâncimile („source_depth_km”) unde cutremurul a erupt sunt măsurate în km. În funcție de rețeaua seismică ce oferă metadatele, adâncimea poate fi relativă la geoidul WGS84, valoarea medie a nivelului mării sau altitudinea medie a stațiilor seismice care au furnizat date despre ora de sosire pentru localizarea cutremurului.

Magnitudinea este aproximativ legată de energia seismică degajată și furnizează un estimat al dimensiunii relative sau al puterii unui cutremur. Există diverse metode (scări) pentru măsurarea magnitudinii. STEAD conține seismograme asociate cu o gamă largă de dimensiuni a cutremurelor, cu magnitudinii cuprinse între -0.5 până la 7.9, dar cutremurele mici (magnitudini < 2.5) compun majoritatea setului de date. Magnitudinile au fost raportate în 23 de scări diferite, unde magnitudinea locală (ml) și magnitudinea de durată (md) sunt majoritare.

3.3. Crearea setului de date personalizat

Mai întâi, a fost necesar un studiu prealabil al documentației setului de date și al modului de structurare al datelor în interiorul fișierelor furnizate pentru a decide modalitatea de alegere a datelor aferente unui singur senzor, aflat într-o locație specifică de pe glob.

Rezultatul studiului documentației setului de date și a altor surse externe precum site-ul IRIS[18] și site-ul QVSData[19][19] a relevat faptul că aparatele seismograf pot fi dotate cu mai multe tipuri de seismometre, instrument de bază cu care se realizează înregistrarea undelor seismice. Cele mai des întâlnite tipuri de seismometre în practică sunt cele de tip HH (High Broadband - High Gain) și cele de tip EH (Extremely Short Period - High Gain). În cadrul acestor denumiri, prima literă specifică rata de eșantionare, iar cea de a doua familia de senzori de care aparține.

Seismometrele de tip HH (ro. bandă largă – amplificare înaltă) au în structură un pendul inerțial cu un mecanism de feedback al forței. Astfel, la detectarea unei activități seismice, se va genera un semnal electric ce va permite stocarea digitală a undelor seismice. Aceste seismometre au o sensibilitate ridicată pentru un interval dinamic foarte larg și pot fi utilizate pentru a înregistra o gamă foarte largă de semnale.[19]

Cu toate aceste, seismometrele de tip EH (ro. perioadă extrem de scurtă – amplificare înaltă) sunt în continuare des utilizate datorită preciziei ridicate și a frecvenței de rezonanță mult mai ridicată decât frecvența majorității undelor seismice. Acest lucru este datorat perioadei naturale foarte scurte de oscilație a pendulului ce permite înregistrarea cutremurelor de magnitudini mări fără a intra în rezonanță. [19]

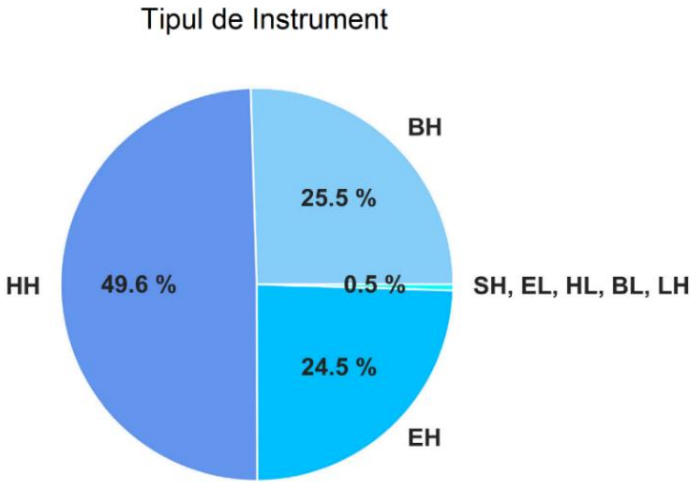


Figura 3.3 Tipuri de instrumente folosite în construirea setului de date STEAD; Sursa: [3]

Informațiile acumulate în studiul menționat au condus la decizia de a alege înregistrările atât ale unui seismometru de tip HH, cât și unul de tip EH, ce se află la aceeași locație, în aceeași rețea și în cadrul aceluiași receptor (cu același cod receptor). Această decizie a fost luată în vederea diversificării viitorului set de date atât cu forme de undă seismice ale unor cutremure de magnitudini mari (peste 3 grade pe scara Richter) oferite de către un seismometru de tip EH, cât și mici.

Prin alegerea datelor oferite de către două seismometre aflate la aceeași locație (același „receiver_latitude”, „receiver_longitude” și „receiver_elevation_m”), în aceeași rețea („network_code”) și în cadrul aceluiași receptor („reciever_code”) va fi redusă apariția erorilor ce ar putea afecta procesul de instruire al rețelei neurale din cauza variației mari a datelor înregistrate de

către receptoare calibrate diferit. Menținerea tuturor datelor în interiorul unei singure zone geografice și asigurarea provenienței acestora din cadrul unei singure stații va conduce și la complexitatea redusă pe care o urmăream.

Următorul pas este implementarea și rularea de script-uri Python asupra unei bucăți (en. chunk) din setul de date ce conține 200,000 de seismograme.

Primul script constă în încărcarea formelor de undă și a etichetelor, afișarea unui număr restrâns de forme de undă cu ajutorul bibliotecii Matplotlib și a informațiilor aferente fiecărei forme de undă. Pentru încărcarea datelor seismice conținute în fișierul în format „.hdf5" și a etichetelor conținute în fișierul „.csv" au fost utilizate bibliotecile h5py și Pandas. Acest script a avut drept scop determinarea modului în care setul de date poate fi încărcat și prelucrat prin executarea unor operații simple de afișare. Rezultatele obținute în urma afișării datelor sunt de forma Figura 3.2.

Plecând de la script-ul anterior, următorul pas este implementarea unui Dataloader personalizat conform structurii bibliotecii PyTorch, care să permită executarea operațiilor de încărcare a setului de date și a etichetelor, aplicarea Transformatei Fourier de Timp Scurt asupra seismogramelor, și întoarcerea spectrogramelor și a etichetelor aferente acestora ca date de ieșire. Spectrogramele sunt calculate cu ajutorul bibliotecii open-source Librosa și vor servi drept date de intrare în rețeaua neurală adâncă, iar etichetele vor fi utilizate în scopul instruirii supervizate a rețelei neurale adânci.[Anexa 2 - 1]

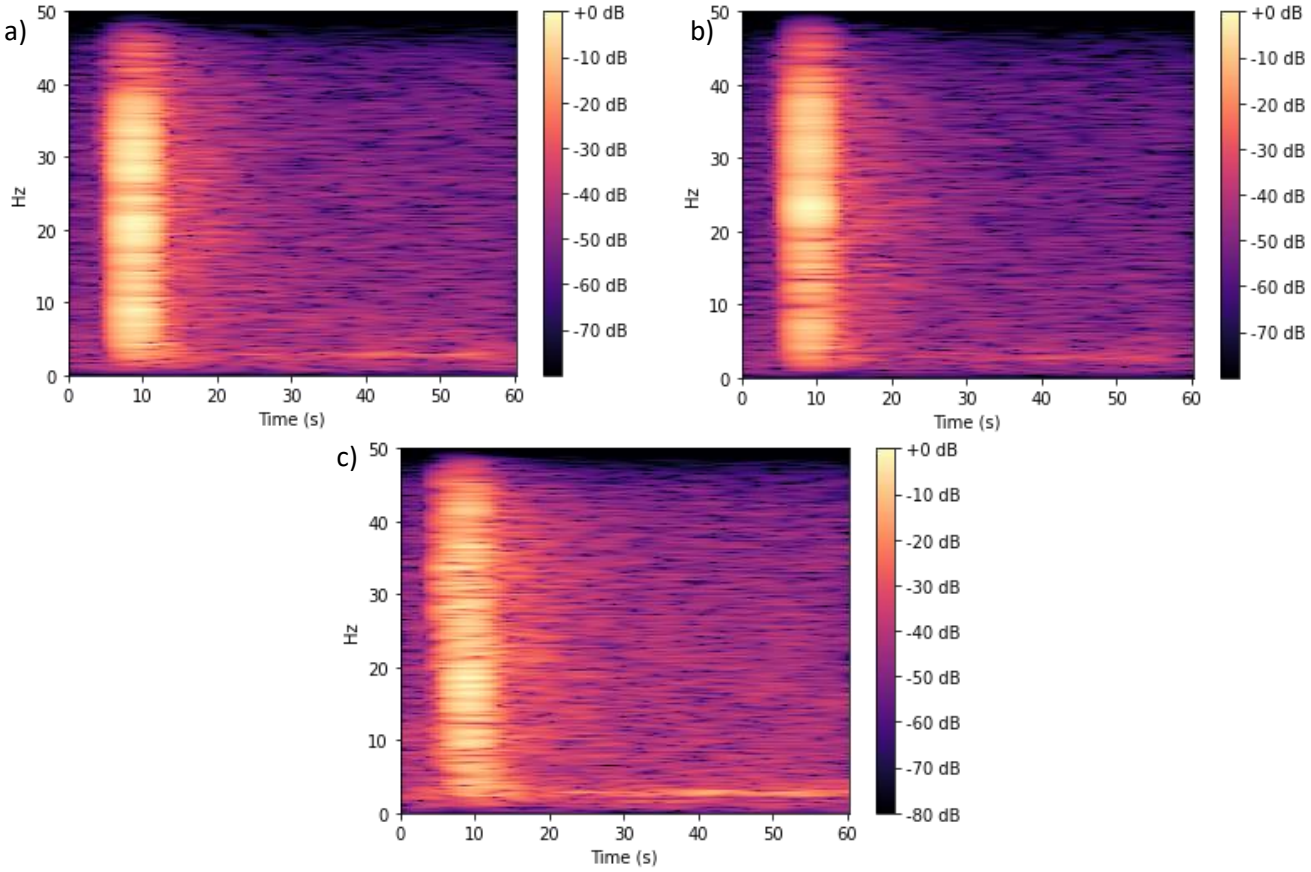


Figura 3.4 Spectrogramele unui seism pe orientările a) Est-Vest, b) Nord-Sud, c) Verticală

Într-un nou script, Dataloader-ul personalizat este folosit pentru parcurgerea setului de date și contorizarea numărului de seismograme înregistrate de către fiecare seismometru. Pentru aceasta, se vor utiliza codul de rețea, codul seismometrului, latitudinea, longitudinea și tipul de seismometru din fișierul de etichete pentru a identifica și diferența seismometrele din întregul set de date. Acestea vor

fi salvate într-un dicționar ce conține drept cheie etichetele menționate anterior, concatenate într-un șir de caractere, iar ca valoare numărul de seismograme. La final, toate cheile și valorile vor fi salvate într-un fișier text, în ordine descrescătoare a valorii, și manual vor fi păstrate doar seismometrele ce conțin peste 10000 de seismograme înregistrate.

Din lista de seismometre ce conțin peste 10000 de înregistrări, am ales locația cu cele mai multe date seismice, atât pentru seismometrul de tip HH, cât și cel EH. Rezultatul a fost alegerea seismometrelor HH și EH ce au următoarele identificatoare de rețea și coordonate "PB;B082;33.598182;-116.596005;1374.8".

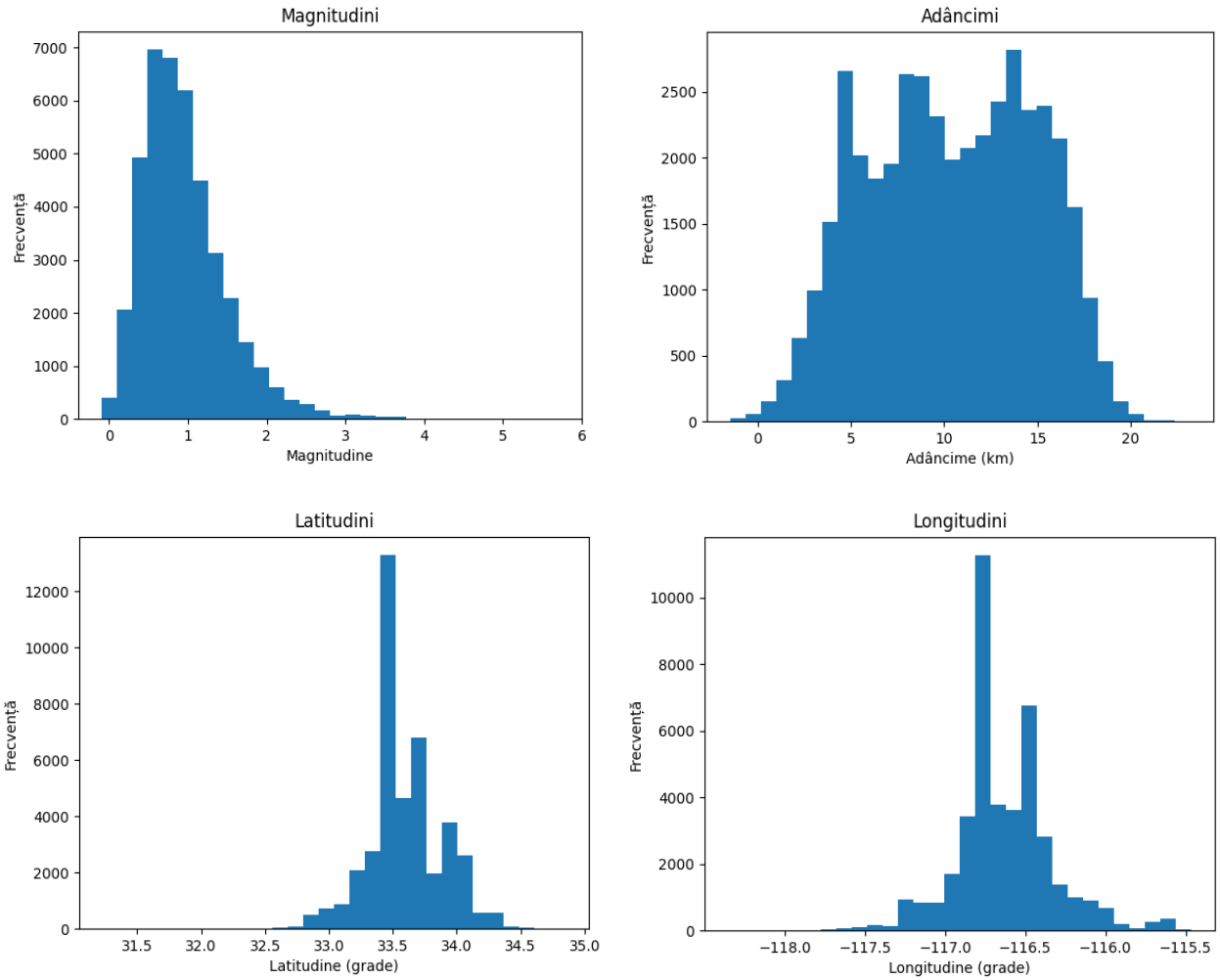


Figura 3.5 Distribuția valorilor caracteristicilor semnalelor seismice în noul set de date

Seismometrele cu codul de rețea PB (Plate Boundary Observatory Borehole Seismic Network) și codul de rețea B082 aferente locației Anza, California, SUA pun la dispoziție un total de 41365 de forme de undă seismice etichetate. Magnitudinile sunt cuprinse între [-0.1, 3.2] grade pe scara Richter pentru seismometrul de tip HH și [0.0, 5.71] grade pe scara Richter pentru seismometrul de tip EH. Tipul de magnitudine înregistrat este local (ml).

Făcută această alegere, setul de date rezultat este suficient vast și de diversificat pentru aplicația propusă. Acesta conține o gamă largă de tipuri de seisme înregistrate cu o gamă a magnitudinilor cuprinsă între [-0.1, 5.71] grade pe scara Richter.

Un nou script va fi utilizat pentru parcurgerea bucății 2 a setului de date STEAD pentru a extrage formele de undă seismice și etichetele aferente seismometrelor alese și a le stoca în noi fișiere

„.hdf5” și „.csv”. Acest noi fișiere ce vor conține seismogramele și etichetele vor reprezenta noul set de date utilizat în instruirea rețelei neurale adânci.

Diversitatea setului de date creat și numărul considerabil de evenimente înregistrate ar trebui să conducă la o generalizare suficient de bună a problemei analizei semnalelor seismice de către rețeaua neurală utilizată, cel puțin pentru zona geografică aleasă.

3.4. Preprocesarea datelor de intrare

Dataloader-ul personalizat este o clasă ce moștenește clasa „Dataset” a bibliotecii PyTorch și implementează un constructor cu parametrii, o metodă pentru întoarcerea dimensiunii setului de date, o metodă pentru extragerea unei (sau unor) date seismice sub formă de spectrogramă împreună cu etichetele aferente și o metodă de calcul a spectrogramelor seismogramei.

Constructorul cu parametrii primește drept parametrii calea către fișierul „.csv” cu etichetele, calea către fișierul „.hdf5” cu formele de undă și o metodă „transform” ce efectuează transformarea datelor de intrare în formatul dorit. În interiorul constructorului sunt inițializate atributele ce stochează etichetele, calea către setul de date, lista denumirilor cutremurelor (trace_name) și transformarea.

Metoda pentru extragerea unei (sau unor) date seismice sub formă de spectrogramă împreună cu etichetele aferente primește ca parametru un index (sau un Tensor de indecși). Pe baza indexului (sau Tensor-ului de indecși convertit în listă) se încarcă setul de date și numele cutremurului (sau cutremurelor), se stochează într-un NumPy array seismogramele, iar apoi se calculează spectrogramele acestora. La final, în urma aplicării transformării asupra datelor de ieșire (dacă este cazul), acestea sunt întoarse sub forma unui dicționar ce conține spectrograme și etichete.

Conform clasei ToTensor [Anexa 2 - 1] definite de către mine, la apelul acestei clase se va realiza prelucrarea (transformarea) datelor de ieșire în felul următor. Se verifică dacă câmpul adâncimii conține o valoarea „None” (ro. nici una) (este singurul câmp din cele necesare care ar putea conține o astfel de valoare). În caz afirmativ se întoarce valoarea „None” ca dată de ieșire, altfel spectrogramele și etichetele sunt stocate în două NumPy arrays diferite, iar acestea la rândul lor sunt convertite în Tensori PyTorch și stocate ca valori într-un dicționar întors de către metodă înapoi Dataloader-ului.

Metoda ce calculează spectrogramele seismogramelor utilizează funcțiile „librosa.stft” și „librosa.amplitude_to_db” ale bibliotecii Librosa deja menționate. Metoda primește ca parametru formele de undă, le separă în trei variabile diferite în funcție de direcția pe care au fost înregistrate, calculează STFT pe fiecare direcție, convertește rezultatele din valori de amplitudine în logaritmic (dB) păstrând pentru această operație doar partea reală (modulul) a STFT, concatenează rezultatele într-un NumPy array și întoarce spectrogramele.

Funcția „librosa.stft” calculează Transformata Fourier de Timp Scurt conform celor prezentate în secțiunea 2.1. Aceasta primește drept parametrii semnalul de intrare, dimensiunea semnalului ferestruit după bordarea cu zero, lungimea saltului, lungimea ferestrei, tipul ferestrei și alți parametrii utili în formatarea rezultatului dorit.

În cazul dimensiunii semnalului ferestruit și bordat, numărul de linii al spectrogramei rezultate va fi de (1 + n_fft/2), unde n_fft e dimensiunea specificată. Dimensiunea standard utilizată de bibliotecă este de 2048 de eșantioane ce corespunde unei durate de 93 ms la o rată de eșantionare de 22050 Hz (rata de eșantionare implicită în librosa). În lucrarea de față am ales o dimensiune de 1024 de eșantioane ținând cont că rata de eșantionare a cutremurelor este mult mai mică decât în cazul

semnalelor audio. Astfel rezoluția în domeniul frecvență va fi suficient de mare, fără a compromite rezoluția temporală a spectrogramei.

Lungimea saltului reprezintă numărul de eșantioane adiacente ce sunt sărite în procesul de calcul al STFT. Cu cât valoarea saltului este mai mică, numărul de coloane va crește, fără a afecta rezoluția spectrală. Dacă este nespecificată, valoarea implicită a saltului este win_length // 4 (împărțirea întreagă a lungimii ferestrei la 4). Un număr mare de coloane în spectrogramă va conduce la o rezoluție mare în domeniul timp, însă operația de calcul a STFT va deveni computațional complexă, crescând și cerințele de memorie.

Lungimea ferestrei este dimensiunea ferestrei de calcul a STFT ce este bordată apoi cu valori de zero astfel încât să se potrivească cu dimensiunea STFT (n_fft). Valorile mici îmbunătățesc rezoluția temporală a STFT (adică capacitatea de a discrimina impulsuri ce sunt strâns distanțate în timp) în detrimentul rezoluției în domeniul frecvență (adică capacitatea de a discrimina tonurile pure care sunt strâns distanțate în frecvență). Acest efect este cunoscut sub numele de compromis de localizare timp-frecvență și trebuie ajustat în funcție de proprietățile semnalului de intrare.[13]

Pentru ca semnalul de intrare să poată fi prelucrat de către toate straturile rețelei convoluționale adânci propuse de mine, cu o rezoluție suficient de bună atât în domeniul timp cât și în frecvență, am ales o dimensiune a semnalului ferestruit pentru calculul STFT de 1024 de eșantioane și o lungime a saltului de 16 eșantioane. Spectrograma rezultată va avea dimensiunile [513, 188] pentru numărul de linii, respectiv de coloane.

În sarcinile de analiză a semnalelor seismice, este importantă surprinderea în mod eficient atât a caracteristicilor temporale, cât și a celor spectrale. Tonurile apropiate sau impulsurile semnalelor seismice pot avea schimbări rapide atât în domeniul timp, cât și frecvență. O rezoluție echilibrată timp-frecvență este soluția optimă pentru discriminarea cu o precizie suficient de bună a acestor caracteristici. Astfel, rezoluțiile finale obținute pentru spectrogramele semnalelor seismice vor fi suficient de mari pentru a realiza cu succes analiza semnalelor seismice de către rețeaua neural adâncă.

Funcția „librosa.amplitude_to_db” primește ca parametru partea reală (modulul) a rezultatului STFT și modul de normare al rezultatului (valoarea maximă). Rezultatul acestei operații este spectrograma convertită în scară logaritmică (dB).

Spectrogramele rezultate și concatenate într-un NumPy array, vor fi convertite într-un Tensor de dimensiunile [3, 513, 188], unde 3 este numărul de spectrograme (pe cele trei direcții), iar celelalte dimensiuni sunt dimensiunile spectrogramelor. Tensorul va servi drept dată de intrare în rețeaua neurală adâncă pentru a executa procesul de analiză al semnalului seismic.

Înainte creării lotului de date de intrare în rețea, o funcție personalizată de „alăturare” (en. collate) [Anexa 3 - 2] elimină din lista datelor de intrare datele ce au valoarea „None”, iar datele rămase sunt alăturare cu ajutorul unei funcții implicite ale bibliotecii PyTorch și sunt întoarse de către Dataloader pentru efectuarea antrenării și testării rețelei neurale adânci. În cazul particular în care toate datele de intrare au valoarea „None”, în locul efectuării alăturării se va întoarce direct valoarea „None”. Când buclele de antrenare, validare și testare vor identifica această situație, iterația respectivă este omisă.

Capitolul 4. Experimente

4.1. Soluția Cloud – Google Colab

Google Colab (Colaboratory) este o soluție cloud oferită de către Google Research ce permite oricărui individ să scrie și execute cod Python prin intermediul browser-ului. Acest produs este în special dedicat aplicațiilor de Machine Learning, analiza datelor sau scopurilor educaționale. Din punct de vedere tehnic, Colab este un serviciu Jupyter Notebook ce nu necesită vreo configurare prealabilă, ce asigură acces gratuit la resurse computaționale, inclusiv resurse GPU. Însă toate aceste beneficii vin cu o limitare, timpul de rulare (en. runtime) pe resursele GPU este limitat, lucru care face necesară antrenarea rețelelor neurale în mai multe etape.

Am ales utilizarea platformei Google Colab în implementarea modelelor rețelelor neurale, în antrenarea și testarea acestora. În ciuda limitărilor aduse de către versiunea gratuită, resursele GPU, memoria RAM și procesoarele puse la dispoziție de către platformă sunt mult mai performante decât sistemele de calcul pe care le dețin personal. O sesiune gratuită de Google Colab cu acces la resurse GPU îmi oferă puterea de calcul a unui sistem cu 12.7 GB de memorie RAM, un GPU NVIDIA Tesla T4 cu 15 GB de memorie și 78.2 GB de memorie de stocare pe Disk (la data de 15.06.2023). Aceste resurse sunt suficiente pentru realizarea experimentelor, însă timpul va fi limitat la aproximativ 6 ore de runtime pe zi.

Utilizarea soluției oferite de către Google va presupune încărcarea setului de date împreună cu etichetele în contul personal de Google Drive și atașarea (en. mount) spațiului de stocare la fiecare rulare a notebook-ului în Colab.[Anexa 3 - 1] Acest lucru va permite și salvarea rezultatelor intermediare sau finale ale antrenării și testării în spațiul de stocare personal al Google Drive.

4.2. Împărțirea setului de date

Setul de date este împărțit în 70% date de antrenare, 15% date de validare și 15% date de testare. Dimensiunea setului de antrenare este aleasă cu un procent dominant față de celelalte seturi pentru a asigura o diversitate suficient de mare de exemple în cadrul procesului de instruire și a conduce la o generalizare cât mai bună a rețelei neurale. În același timp, trebuie păstrat un număr suficient de mare și de exemple pentru validarea, respectiv testarea finală a rețelei pe un lot de date neutilizat în procesul de instruire. Împărțirea setului de date în 70% date de antrenare – 30% date de testare este o bună practică frecvent adoptată în cadrul implementării și instruirii rețelelor neurale cu învățare supervizată.

Împărțirea datelor în date de antrenare, validare și testare este făcută în mod aleator, însă această generare este făcută controlat, pentru a asigura reproductibilitatea ulterioară a experimentelor efectuate.[Anexa 3 - 3] Controlul împărțirii setului de date este realizat de utilizarea unui „seed” (în cazul meu, 19) ce va fi aplicat în calculul împărțirii setului.

Pe lângă setul de antrenare și cel de testare, ce sunt utilizate în antrenarea, respectiv testarea ulterioară antrenării a performanțelor rețelei neurale adânci, în unele situații este utilizat și un set de validare. Setul de validare este un set adițional ce nu se folosește în instruirea rețelei, dar care are rolul de a valida noile caracteristici învățate în cadrul epocii de antrenare în desfășurare. Astfel, dacă

funcția de cost scade atât pe setul de antrenare, cât și pe cel de validare, se poate spune că rețeaua neurală a căpătat noi cunoștințe și abilități în rezolvarea problemei.

Evoluția ideală a procesului de antrenare este aceea când valoarea funcției de cost pe setul de validare este mai mică decât cea pe setul de antrenare. Într-o astfel de situație se poate spune că rețeaua neurală generalizează în mod corect soluția rezolvării problemei, fiind capabilă să rezolve și probleme neîntâlnite în timpul instruirii. Însă dacă valoarea funcției de cost pe setul de validarea este mai mare decât cea pe setul de antrenare, atunci rețeaua nu generalizează suficient soluția problemei.

Există două cazuri particulare în care valoarea funcției de cost pe setul de antrenare și validare nu evoluează conform cerințelor. Situația în care costul pe setul de validare este mult mai mare decât cel pe setul de antrenare și situația în care ambele costuri sunt foarte mari sunt cele două cazuri negative des întâlnite în procesul de antrenare al rețelelor neurale.

În prima situație, rețeaua neurală este mult prea complexă pentru sarcina dată și învață, astfel, „pe de rost” setul de date de antrenare datorită capacității mari de memorare (memorizează soluțiile setului de date). În această situație se ajunge cel mai adesea în urma antrenării excesive a rețelei neurale adânci, pentru un număr mult mai mare de epoci decât cel minim necesar pentru ca rețeaua să poată generaliza soluția sarcinii. Pentru a evita această situație fie se folosesc metode de regularizare precum cele menționate în secțiunea 2.2, fie se alege un set date mai mare sau se extinde setul de date actual prin metode de augmentare, dacă aplicația permite aceste lucruri.

În a doua situație, rețeaua neurală se află în cazul complementar. Arhitectura acesteia este mult prea simplă pentru sarcina dată și nu reușește să găsească o soluție pentru a corela datele de intrare cu etichetele acestora. Pentru a putea depăși această situație fie se antrenează rețeaua neurală pentru mai mult timp, fie se alege o arhitectură neurală cu o capacitate mai mare de învățare în cazul în care antrenarea pe o perioadă de timp mai îndelungată nu este suficientă.

Împărțirea setului de date de 41365 de evenimente seismice va fi realizată în felul următor:

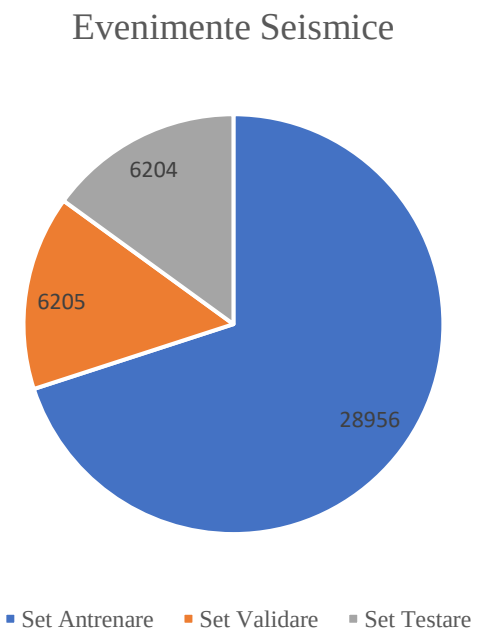


Figura 4.1 Împărțirea setului de date

4.3. Procesul de antrenare al rețelelor neurale adânci

Rețelele neurale adânci implementate sunt cele prezentate în Capitolul 2. Modelele sunt implementate atât sub forma arhitecturii LeNet (fără salturi de conexiune)[Anexa 1 - 1], cât și a arhitecturii ResNet[Anexa 1 - 2] cu blocurile convoluționale și reziduale definite. Modelul ResNet este implementat, antrenat și testat în 2 arhitecturi diferite, una conform Tabel 2.1, iar alta cu un bloc rezidual adițional adăugat conform Anexa 1 - 3.

Experimentele derulate utilizează o dimensiune a lotului de date de intrare (en. batch size) de 32 de semnale seismice și un număr maxim de 50 de epoci de antrenare. Rata de învățare inițială este aleasă în funcție de rețeaua neurală antrenată, în funcție de complexitatea și capacitatea de învățare a acesteia.

În procesul de învățare, pe lângă funcția de cost eroare pătratică medie și optimizatorul Adam, prezentate în cadrul paragrafului 1.3.2, se va mai folosi un programator al ratei de învățare (en. learning rate scheduler) ce va ajusta rata de învățare atunci când nu sunt detectate îmbunătățiri ale costului pe setul de validare. Programatorul ratei de învățare ReduceLROnPlateau (ro. reduce rata de învățare pe platou) va asigura că antrenarea nu se va opri prematur, în regiunile sub formă de platou ale funcției de cost, prin ajustarea ratei de învățare cu un factor de 0.1 atunci când performanța nu se îmbunătățește pentru un număr de epoci stabilit. Procesul de antrenare va continua, astfel, până la găsirea soluției celei mai apropiate de cea optimă global. [Anexa 3 - 4]

Metrica de performanță utilizată pentru determinarea costului rețelei neurale adânci în procesul de estimare al parametrilor surselor seismice este, în cazul lucrării de față, eroarea pătratică medie (en. Mean Square Error). În statistică, eroarea (sau abaterea) pătratică medie a unui estimator măsoară media erorilor pătrate, adică diferența medie pătratică între valorile estimate și valorile reale. MSE este o măsură a calității unui estimator, iar pentru că este derivată din pătratul distanței Euclidiene, are întotdeauna o valoare pozitivă ce scade pe măsură ce eroarea se apropie de zero.[20]

Antrenarea rețelei neurale adânci în condițiile unui timp limitat de rulare oferit în mod gratuit de către Google pentru sesiunile cu acces la resurse GPU va presupune salvarea stării la fiecare epocă de antrenare (en. checkpoint) pentru a putea relua ulterior procesul de antrenare când resursele vor redeveni disponibile. Salvarea stării procesului de antrenare presupune salvarea într-un dicționar a numărului epocii la care s-a realizat ultima salvare, salvarea dicționarului de stare al modelului, al optimizatorului, al programatorului ratei de învățare și salvare istoricului valorilor de cost în cadrul epocii respective. Dicționarul obținut este salvat în spațiul personal de stocare Google Drive într-un format specific bibliotecii PyTorch, având extensia „.pt”, cu ajutorul instrucțiunii „torch.save(model, PATH)”.[Anexa 2 - 3] Atunci când se dorește reluarea instruirii rețelei neurale, dicționarul este încărcat din Google Drive cu instrucțiunea „torch.load(PATH)”, iar fiecare dicționar de stare este încărcat în obiectul corespunzătoare cu ajutorul unei instrucțiuni de încărcare, precum în Anexa 3 - 4.

Procesul de antrenare presupune în prima etapă preluarea unui lot de 32 de seismograme, trecerea lor pe dispozitivul utilizat (în cazul meu GPU) și calcularea estimărilor ieșirilor pentru fiecare seismogramă. În a doua etapă, pe baza estimărilor rezultate și a valorilor observate (reale) se calculează MSE, iar în funcție de valoarea obținută pentru MSE se vor ajusta ponderile rețelei prin algoritmul de optimizare ales.

În bucla de antrenare și validare sunt cumulate costurile înregistrate pe parcursul epocii de antrenare pentru a putea calcula, la final, costul mediu (în cazul meu, eroarea pătratică medie) înregistrată pe ambele seturi în cadrul epocii respective. Costurile medii ale epocii de antrenare servesc drept metrică de performanță a rețelei neurale și ajută la monitorizarea capacității rețelei de a rezolva problema dată și a găsi o soluție în spațiul datelor de intrare care să conducă la ieșirile dorite.

Pentru a evita situația în care rețeaua intră în supradaptare (en. overfitting) după un număr mare de epoci de antrenare, se introduce o metodă de regularizare (en. regularization) denumită „early stopping” (ro. oprire timpurie). Regularizarea este un proces ce ajustează soluțiile rezultate în cazul unor probleme de Machine Learning pentru a fi mai „simple” și a evita supradaptarea (sau supraînvățarea). Metoda de „early stopping” are rolul de a opri procesul de antrenare atunci când pentru un număr stabilit de epoci de antrenare nu se observă nicio îmbunătățire semnificativă a performanțelor sistemului.

Metoda de „early stopping” este implementată sub formă de clasă, ce conține un constructor cu parametrii, getteri și setteri pentru atributele ce se modifică în timp sau necesită preluarea lor pentru alte operații, metode de salvare și încărcare în Drive a valorilor costurilor din cadrul epoci cele mai performante și metoda de „earlyStop” ce are rolul de a verifica dacă performanțele rețelei s-au îmbunătățit sau nu în cadrul epocii în curs.

În cazul în care performanțele rețelei s-au îmbunătățit, atunci ponderile rețelei sunt salvate în Drive în format „.pt” prin instrucțiunea „torch.save”, costurile sunt salvate pentru comparații ulterioare, numărul epocii este salvat, iar contorul ce reține numărul de epoci consecutive în care nu s-au detectat îmbunătățiri este resetat.

În cazul lucrării de față, numărul de epoci așteptate consecutiv până la oprirea procesului de învățare este 4. Acest număr a fost ales empiric, prin executarea mai multor experimente și observarea numărului maxim de epoci consecutive după care s-a detectat din nou o îmbunătățire semnificativă.[Anexa 2 - 2] Astfel se va economisi timp de rulare și nu se va antrena excesiv rețeaua neurală, obținând rezultatele optime pentru starea ponderilor rețelei.

Modalitatea de analiză a performanțelor pe setul de antrenare și cel de validare va consta în studierea evoluției erorii pătratice medii, în cazul fiecărui parametru în parte, în funcție de numărul epocii de antrenare. În urma finalizării procesului de antrenare, pe baza istoricului abaterilor pătratice medii stocate în dicționarul menționat anterior, se va trasa graficul valorilor abaterilor pătratice medii în funcție de epoca de antrenare.[Anexa 2 - 3] Acest grafic va afișa care a fost progresul procesului de antrenare al rețelei neurale și cât de bine a fost îndeplinită această sarcină.

4.4. Procesul de testare al rețelelor neurale adânci

Procesul de testare al performanțelor rețelei neurale adânci se realizează în aceeași manieră precum antrenarea acesteia, însă printr-o singură iterație a întregului set de testare. La intrarea rețelei neurale sunt prezentate loturi de date de testare, sunt calculate predicțiile, pe baza acestora sunt determinare abaterile pătratice medii de la valorile observate (ideale), iar la final, suma tuturor erorilor este mediată la numărul de iterații efectuate prin set.

Etapa de testare a rețelei neurale vine cu nouă metrică de performanță, coeficientul de determinare R^2. În statistică, coeficientul de determinare reprezintă proporția de variație a variabilei dependente ce este estimată pe baza unei (sau unor) variabile independente. Este o statistică utilizată în contextul modelelor statistice ce au drept rol principal fie estimarea (en. prediction) unor rezultate viitoare, fie testarea unor ipoteze, având la bază alte informații conexe. În domeniul învățării automate, coeficientul de determinare oferă o măsură a cât de bine sunt reproduse (estimate) rezultatele observate de către modelul statistic reprodus.[21]

Cea mai mică valoare posibilă a lui R^2 este 0, iar cea mai mare valoare posibilă este 1. Dacă modelul are o valoare a coeficientului de determinare egală cu 0, atunci acesta nu reușește să estimeze rezultatul dorit. În schimb, dacă modelul are o valoare a R^2 între 0 și 1, modelul reușește estimarea parțială a rezultatului, iar dacă valoarea coeficientului este cât mai apropiată de 1, atunci acesta

produce o estimare a rezultatului cât mai apropiată de cea ideală (scorul $R^2 = 1$ reprezentând situația estimării perfecte a rezultatului).

Există, însă, și cazuri în care R^2 poate avea valori negative. Aceste cazuri apar cel mai des atunci când estimările comparate cu rezultatele ideale dorite nu au fost derivate dintr-o procedură de adaptare a modelului folosind aceste date. În cazul lucrării de față, chiar dacă a fost folosită o procedură de adaptare a modelului folosind date ale aceluiași set, R^2 poate fi totuși negativ din pricina datelor aferente setului de testare, conform ipotezei de mai sus, dar și a din pricina utilizării unei funcții neliniare pentru a potrivi datele. În unele cazuri, dacă rețeaua neurală nu este antrenată suficient sau dacă datele setului de testare nu sunt corelate cu cele ale setului de antrenare, R^2 poate obține valori negative. În cazurile în care apar valori negative, media datelor oferă o estimare mai bună a rezultatelor decât valorile funcției ajustate, conform acestui criteriu.

O altă metodă de evaluare a potrivirii valorilor estimate (Y_estimat) cu cele reale (Y_observat) este reprezentarea într-o diagramă de dispersie (en. scatter plot) a intersecției valorilor observate, de pe abscisă, cu valorile estimate, de pe ordonată. Astfel, dacă corelația dintre valorile estimate și cele observate există și este suficient de mare, punctele se vor grupa în lungul unei drepte de regresie cu o anumită dispersie. Dacă punctele nu se vor grupa în lungul unei drepte, atunci acestea arată faptul că valorile estimate și observate sunt slab dependente statistic. Dacă punctele se situează în lungul unei drepte, dar dispersia lor este mare, atunci înseamnă că în interiorul datelor există o abatere pătratică medie.

În timpul procesului de testare, pentru fiecare lot de date de intrare sunt calculate abaterile pătratice medii ale fiecărui parametru al sursei de semnal seismic, este actualizată valoarea coeficientului de determinare R^2 și sunt salvate într-un dicționar perechile de valori reale – valori estimate obținute. La finalul procesului de testare al rețelei, se va putea calcula eroarea pătratică medie și scorul R^2 pe întregul set de date precum și reprezentarea grafică a dispersiei punctelor valori reale – valori estimate.[Anexa 2 - 4]

4.5. Experiment LeNet

Primul experiment executat a fost antrenarea modelului de tip LeNet[Anexa 1 - 1] în vederea estimării magnitudinii, latitudinii, longitudinii și adâncimii surselor seismice ce au avut loc în zona geografică aleasă: Anza, California. Rata de învățare aleasă în cazul acestei rețele a fost de $5.5*10^{-4}$. Această rată de învățare este mai ridicată comparativ cu cele ale rețelelor ResNet din pricina complexității scăzute a rețelei LeNet. Astfel, procesul de învățare va fi accelerat la început, reducând timpul total necesar antrenării rețelei neurale adânci.

Antrenarea rețelei neurale s-a desfășurat până la depășirea numărului de epoci setat în metoda de regularizare „early stopping”, iar ultima epocă de antrenare a fost epoca 31. Graficele evoluției MSE pe parcursul epocilor de antrenare pentru fiecare caracteristică estimată sunt următoarele:

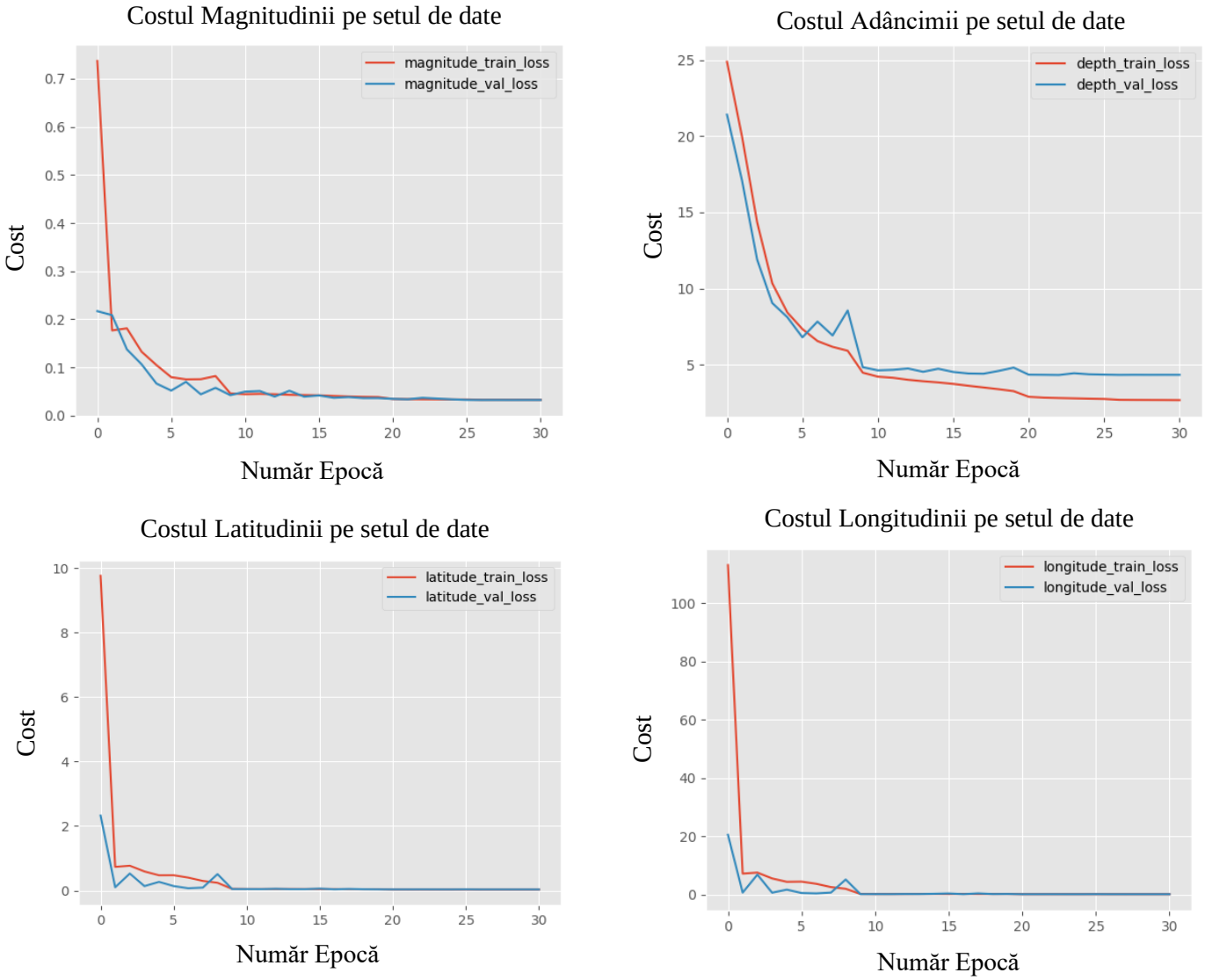


Figura 4.2 Costurile Rețelei Neurale tip LeNet

Ținând cont de metoda de regularizare abordată, „early stopping”, și de faptul că numărul de epoci după care antrenarea se oprește în cazul în care nu se înregistrează îmbunătățiri pe setul de validare este 4, epoca în care s-au obținut parametrii ideali ai rețelei LeNet este epoca 27. O singură epocă de antrenare a durat în medie 15 minute, tot procesul de antrenare durând 465 de minute pentru cele 31 de epoci.

Graficul portocaliu reprezintă evoluția funcției de cost pe setul de antrenare, iar graficul albastru reprezintă evoluția funcției de cost pe setul de validare. Evoluția costului magnitudinii pe setul de antrenare și validare urmează un parcurs „nezgomotos” și fără multe momente de depășire ale costului pe setul de validare față de cel de antrenare. Rețeaua reușește să învețe suficient de bine această caracteristică a sursei seismice, cu o abatere pătratică medie mică pe ambele seturi de antrenare și validare.

În cazul funcției de cost a adâncimii pe setul de validare se poate observa că este puțin mai zgomotoasă decât cea pe setul de antrenare și mult mai zgomotoasă decât în cazul magnitudinii. După epoca 5, eroarea pătratică medie pe setul de validare devine mai mare decât cea pe setul de antrenare, iar aceasta începe scadă mult mai lent decât costul pe setul de antrenare. Acest lucru arată faptul că rețeaua supraînvață problema estimării adâncimii, față de problema magnitudinii.

Atât latitudinea cât și longitudinea au o scădere rapidă a funcției de cost, mult mai rapidă decât în cazul parametrilor menționați anterior, însă erorile pătratice medii sunt mai mari decât în cazul magnitudinii. Latitudinea are o eroare mult mai mică față de longitudine atât pe setul de validare cât și cel de antrenare. Acest lucru demonstrează că longitudinea este cel mai dificil parametru de estimat dintre latitudine, longitudine și magnitudine, iar magnitudinea este parametrul cel mai ușor de învățat și estimat dintre toți cei patru parametrii.

Scăderea rapidă a funcțiilor de cost și valorile inițiale mari pentru funcția de cost pe setul de antrenare a latitudinii și longitudinii sunt determinate de către tipul problemei impuse de către acești parametrii seismici. Sarcina determinării locației unei surse de semnal într-un plan bidimensional necesită cel puțin trei senzori receptori necoliniari pentru a o realiza cu succes. Determinarea latitudinii și longitudinii pe baza informațiilor conținute de spectrograma seismogramei captate de către un singur seismometru va reprezenta o adevărată provocare pentru rețeaua neurală antrenată.

În prima epocă de antrenare, abaterea pătratică medie pe setul de antrenare este foarte mare ca mai apoi pe setul de validare să scadă semnificativ. Atât rețeaua neurală tip LeNet cât și celelalte rețele ResNet vor necesita cel puțin o epocă pentru ajustarea ponderilor rețelei în vederea estimării coordonatelor epicentrelor aflate în zona geografică restrânsă aleasă. Acest comportament indică faptul că rețelele neurale încearcă să învețe estimarea epicentrelor pe baza caracteristicilor spectrogramei de la intrare, însă dificultatea sarcinii și insuficiența informațiilor utile estimării parametrilor face ca rețeaua neurală „să se satureze”. După un număr ridicat de epoci de antrenare, costul pe setul de validare va începe să scadă din ce în ce mai lent, pe când costul pe setul de antrenare va continua să scadă la fel de rapid. Aceste lucruri indică începutul supraadaptării rețelei neurale adânci la problema dată.

Metrică	Antrenare	Validare	Testare
MSE General	0.70137	1.11172	1.07620
MSE Magnitudine	0.03283	0.03244	0.03122
MSE Latitudine	0.03281	0.03493	0.03439
MSE Longitudine	0.04325	0.04592	0.04262
MSE Adâncime	2.69660	4.33358	4.19656

Tabel 4.1 Costurile finale ale rețelei LeNet

În etapa de testare, erorile pătratice medii au fost la fel de mici sau chiar mai mici decât în cazul rezultatelor obținute pe setul de validare sau antrenare, lucru care subliniază faptul că rețeaua a învățat în mod corespunzător extragerea caracteristicilor necesare din datele de intrare pentru estimarea valorilor de ieșire.

Metrică – set Testare	Magnitudine	Latitudine	Longitudine	Adâncime
Scor R^2	0.8862462	0.5661469	0.46937877	0.7915708

Tabel 4.2 Scorul R^2 pe setul de test - LeNet

Valorile coeficientului R^2 pe setul de test sunt promițătoare în cazul magnitudinii și adâncimii, pe când în cazul latitudinii și longitudinii prezintă o corelație destul de slabă a valorilor reale și estimate.

Cu toate că adâncimea are o eroare pătratică medie destul de mare de aproximare, valoarea coeficientului R^2 pe setul de testare este mult mai bună, scoțând în evidență faptul că a reușit să extragă cu o corelație suficient de bună trăsăturile spectrogramelor de intrare.

Valorile mici ale latitudinii și longitudinii pentru R^2, deși abaterile pătratice medii pe setul de test sunt la fel de mici precum cele de pe setul de validare, evidențiază faptul că rețeaua nu a învățat în totalitate să estimeze cu o precizie suficient de ridicată acești parametrii, lipsind foarte mult corelația dintre valorile estimate și cele reale. Acest lucru este datorat în principal de utilizarea unui singur senzor în vederea estimării unui punct într-un plan, o problemă mult prea complexă pentru rețeaua neurală adâncă de tip LeNet implementată. Deși sunt utilizate caracteristici ale formelor de undă seismice în domeniul timp-frecvență, nu sunt suficiente pentru estimarea coordonatelor unui epicentru. Cu toate acestea, faptul că există un scor pozitiv al lui R^2 și că eroarea pătratică medie este suficient de mică demonstrează faptul că rețeaua reușește să învețe câteva din trăsăturile importante în estimarea coordonatele epicentrului unui cutremur.

În urma reprezentării graficului de dispersie al punctelor estimate comparativ cu cele reale (observate) se confirmă valorile obținute pentru coeficientul de determinare R^2 prezentate anterior. Magnitudinea și adâncimea prezintă cea mai mare corelație a punctelor prin poziționarea lor în lungul unei drepte de regresie, însă abaterea pătratică medie a punctelor adâncimii este mult mai mare decât în toate celelalte trei cazurile. Graficele aferente latitudinii și longitudinii evidențiază faptul că eroarea pătratică medie este destul de mare în comparație cu problema ce trebuie rezolvată și că corelația punctelor nu este la fel de puternică precum în cazul magnitudinii, însă aceasta există. De asemenea, faptul că toate valorile observate și prezise pentru latitudine și longitudine se află în același interval de valori confirmă faptul că rețeaua neurală încearcă să învețe caracteristicile necesare și modalitatea de estimare a acestor parametrii ai sursei seismice, însă sarcina este mult prea complexă pentru a o putea generaliza.

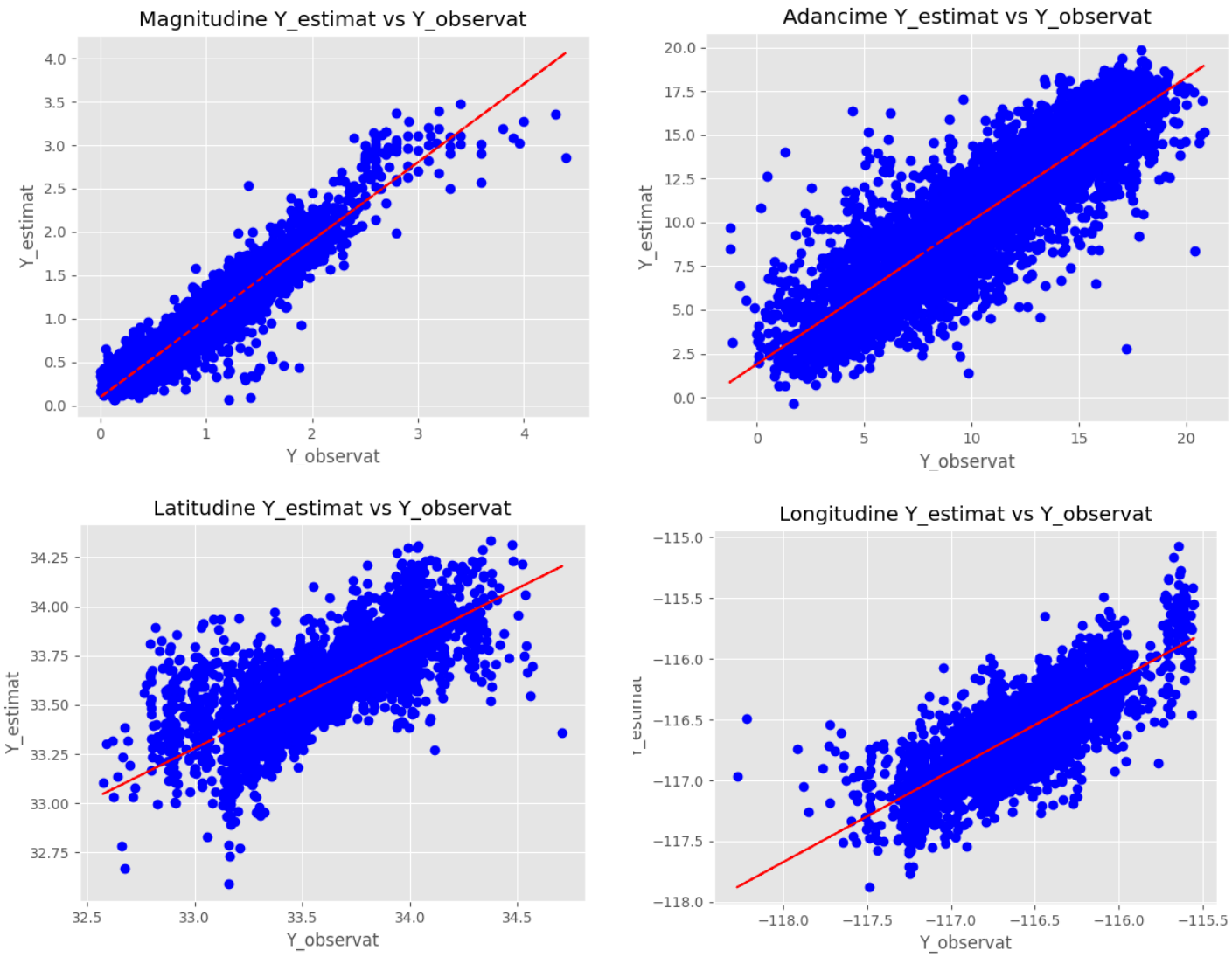


Figura 4.3 Graficul de dispersie al valorilor Y_observat - Y_estimat - LeNet

4.6. Experiment ResNet

Al doilea experiment executat a fost antrenarea modelului de tip ResNet[Anexa 1 - 2] în vederea estimării acelorași parametrii ai unei surse seismice. Rata de învățare aleasă, în cazul acestei rețele a fost de $3*10^{-4}$, o rată mai scăzută comparativ cu cea utilizată în cazul rețelei LeNet pentru a evita supraînvățarea rețelei într-un număr mic de epoci.

Antrenarea rețelei neurale s-a desfășurat până la depășirea numărului de epoci setat în metoda de regularizare, iar ultima epocă de antrenare a fost epoca 16. Graficele evoluției MSE pe parcursul epocilor de antrenare pentru fiecare caracteristică estimată sunt următoarele:

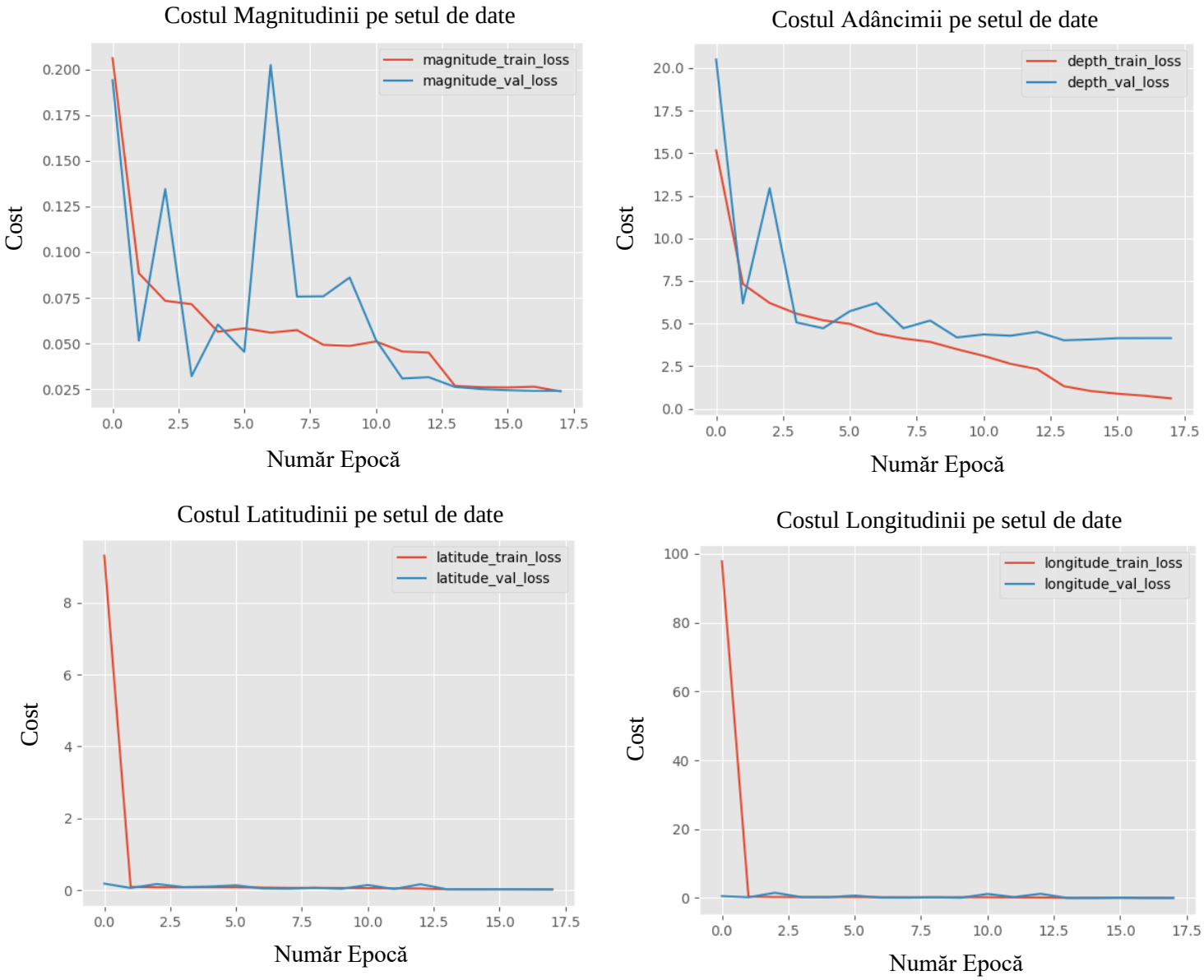


Figura 4.4 Costurile Rețelei Neurale tip ResNet

Durata unei epoci de antrenare în cazul ResNet este în medie de aproximativ 20 minute, tot procesul de antrenare durând 320 de minute pentru cele 16 epoci.

Ținând cont de metoda de regularizare utilizată și de faptul că numărul de epoci după care antrenarea se oprește în cazul în care nu se înregistrează îmbunătățiri pe setul de validare este 4, epoca în care s-au obținut parametrii ideali ai ResNet este epoca 12.

Deși sunt mult mai zgomotoase, graficele funcției de cost pe setul de validare conduc către valori mult mai bune ale metricii de performanță pentru parametrii adâncime și magnitudine față de implementarea rețelei LeNet. Rezultatele mult mai bune sunt datorate capacității de învățare mult mai ridicată a rețelei neurale și a tehnicii de antrenare prin care este întârziată instruirea anumitor straturi convoluționale reziduale datorită prezenței salturilor de conexiune. Acest fapt se reflectă în scăderea exponențială mult mai rapidă a erorii pătratice medii pe setul de antrenare față de cazul LeNet.

Toate aceste lucruri se aplică și în cazul latitudinii și longitudinii, erorile pătratice medii fiind mult mai bune și scăderea mult mai rapidă comparativ cu LeNet. În continuare afirmațiile făcute în cadrul paragrafului anterior legate de motivul valorilor ridicate ale erorii pătratice medii din cadrul primei epoci de antrenare și al scăderii bruște a erorii pe setul de antrenare rămân valabile.

Complexitatea ridicată a ResNet face ca procesul de antrenare să dureze mult mai puțin decât în cazul rețelei neurale tip LeNet, însă timpul de procesare al unei singure date de intrare (latența) crește considerabil. Cu toate acestea, creșterea în latență adusă de către ResNet nu este suficient de mare încât să depășească beneficiile aduse de performanțele sporite, latența crescută resimțindu-se doar în cazul procesării volumelor mari de date seismice.

Forma zgomotoasă a graficelor este cel mai probabil cauzată de capacitatea ridicată de învățare a ResNet-ului influențată în mod negativ de rata de învățarea inițial aleasă puțin mai mare decât valoarea ideală. Cu toate acestea, rata de învățare este ajustată pe parcursul procesului de antrenare atât de către optimizator, în mod indirect, cât și de către programatorul ratei de învățare, în mod direct, conform explicațiilor date în paragraful 4.3. O rețea de complexitate ridicată și cu o capacitate de învățare mare precum ResNet ar trebui să folosească o rată de învățare mai scăzută în cadrul problemelor complexe pentru a evita apariția divergenței în procesul de ajustare a ponderilor.

Valorile finale obținute pentru abaterea pătratică medie pe setul de validare au suferit îmbunătățiri semnificative comparativ cu cele ale LeNet. Magnitudinea, latitudinea și longitudinea au obținut valori de 0.026, 0.023, respectiv 0.034 comparativ cu valorile de 0.032, 0.034 și 0.045 obținute de către LeNet. De asemenea, abaterea pătratică medie a adâncimii a obținut și ea o îmbunătățire aproximativ 0.315 km față de performanța oferită de LeNet.

Metrică	Antrenare	Validare	Testare
MSE General	0.35211	1.02551	1.00401
MSE Magnitudine	0.02685	0.02628	0.02468
MSE Latitudine	0.02054	0.02328	0.02240
MSE Longitudine	0.04006	0.03421	0.03391
MSE Adâncime	1.32098	4.01828	3.93505

Tabel 4.3 Costurile finale ale rețelei ResNet

Pe setul de test, rezultatele sunt mult mai bune atât din punct de vedere al erorii pătratice medii cât și al coeficientului de determinare, față de rețeaua LeNet. ResNet a reușit să învețe într-un timp mult mai scurt și cu o precizie mult mai mare caracteristicile definitorii ale spectrogramelor

semnalelor seismice și modul de estimare al parametrilor sursei de semnal seismic folosind aceste caracteristici.

Etapa de testare a demonstrat faptul că rețeaua neurală a avut mai mult succes în extragerea caracteristicilor surselor seismice fără a supraînvăța setul de instruire. Faptul că eroarea pătratică medie pe setul de testare, în cazul tuturor parametrilor sursei de semnal este mai mică decât cea pe setul de validare confirmă faptul că rețeaua neurală a reușit să generalizeze metoda de analiză a semnalelor seismice ce au loc în zona geografică din California. Deși, în cazul adâncimii, eroarea pătratică medie de estimare mai mică pe setul de antrenare decât pe cel de validare sau testare, scorul R^2 este mult mai bun decât în cazurile latitudinii și longitudinii.

Metrică – set Testare	Magnitudine	Latitudine	Longitudine	Adâncime
Scor R^2	0.9100773	0.71742815	0.5779206	0.8045777

Tabel 4.4 Scorul R^2 pe setul de test - ResNet

Atât latitudine, dar mai ales longitudinea au înregistrat îmbunătățiri semnificative ale erorii pătratice medii și coeficientului de determinare pe setul de testare. Saltul semnificativ făcut de către coeficientul de determinare în cazul latitudinii arată că se poate determina acest parametru cu o precizie destul de bună, însă nu la fel de bună precum cea a magnitudinii. Rețeaua neurală a reușit să extragă noi trăsături care să ajute în estimarea locației epicentrului cutremurului, însă longitudinea continuă să rămână cel mai greu de estimat parametru dintre toate acestea.

Graficul de dispersie al valorilor observate comparativ cu cele estimate evidențiază creșterea obținută în performanță prin scăderea abaterii pătratice medii și creșterea coeficientului de determinare în cazul tuturor parametrilor sursei seismice. Din graficul magnitudinii și adâncimii se poate observa o scădere clară a dispersiei punctelor, iar în cazul latitudinii și longitudinii se poate distinge creșterea în corelație dintre valorile reale și cele estime. Conform rezultatelor, latitudinea este parametrul care a obținut cea mai mare creștere în corelație între valorile reale și cele estimate, comparativ cu rezultatele obținute anterior pentru LeNet.

În urma rezultatelor obținute se poate afirma că rețeaua neurală adâncă de tip ResNet are arhitectura cea mai adecvată pentru analiza semnalelor seismice și estimarea parametrilor sursei de semnal. Cunoscând aceste lucruri, următorul experiment urmărește antrenarea și testarea unei rețele neurale adânci de tip ResNet extinsă cu un bloc rezidual adițional ce va conține 128 de filtre, iar ultimul bloc convoluțional, ce urmează blocului rezidual adăugat, va fi modificat pentru a avea 256 de filtre (sau nuclee). Pentru ca spectrogramele seismogramelor să poată fi prelucrate de către întreaga rețea neurală fără ca dimensiunea acestora să devină mult prea mică, lungimea saltului ferestrei în calculul STFT va fi scăzută de la 32 de eșantioane la 16 eșantioane pentru a asigura o dimensiune temporală suficient de mare. Noile dimensiuni ale Tensorului de intrare vor fi [3, 513, 376], fără ca rezoluția spectrală să fie afectată. Modificând doar lungimea saltului ferestrei de calcul, rezultatele experimentului nu vor fi influențate decât de noua arhitectură a rețelei neurale adânci.

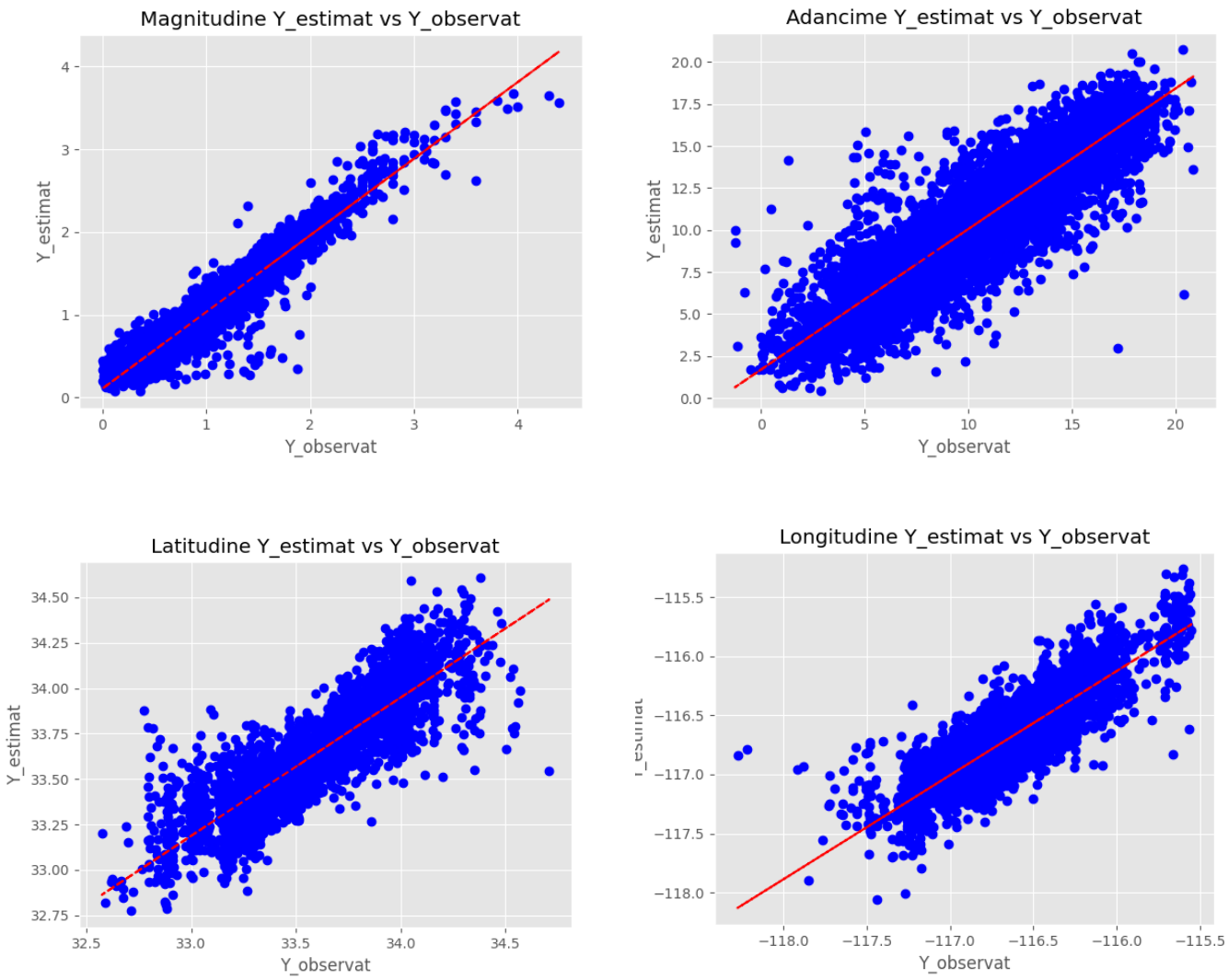


Figura 4.5 Graficul de dispersie al valorilor Y_observat - Y_estimat - ResNet

4.7. Experiment ResNet extins

Implementarea rețelei de tip ResNet extinsă este cea prezentată în Anexa 1 - 3, în care s-a mai adăugat un bloc rezidual adițional, extinzând numărul de filtre disponibile în întreaga rețea. Prin această extindere am urmărit creșterea potențialului rețelei de a extrage trăsături și mai avansate și de a găsi legături și mai complexe în interiorul datelor de intrare.

Scopul experimentului a fost antrenarea modelului de tip ResNet extins în vederea estimării acelorași parametrii ai unei surse seismice. Rata de învățare aleasă, în cazul acestei rețele a fost de $1*10^{-4}$, cea mai scăzută rată de învățare comparativ cu cele utilizate în cazul celorlalte rețele prezentate, pentru a asigura o parcurgerea lentă a spațiului funcției de cost și a obține soluția cea mai apropiată de cea optimă global. În acest fel se va evita și obținerea unor grafice zgomotoase ale evoluției funcției de cost din pricina complexității ridicate a problemei.

Antrenarea rețelei neurale s-a desfășurat până la depășirea numărului de epoci stabilit pentru metoda de regularizare „early stopping”, iar ultima epocă de antrenare a fost epoca 20. Graficele evoluției MSE pe parcursul epocilor de antrenare pentru fiecare caracteristică estimată sunt:

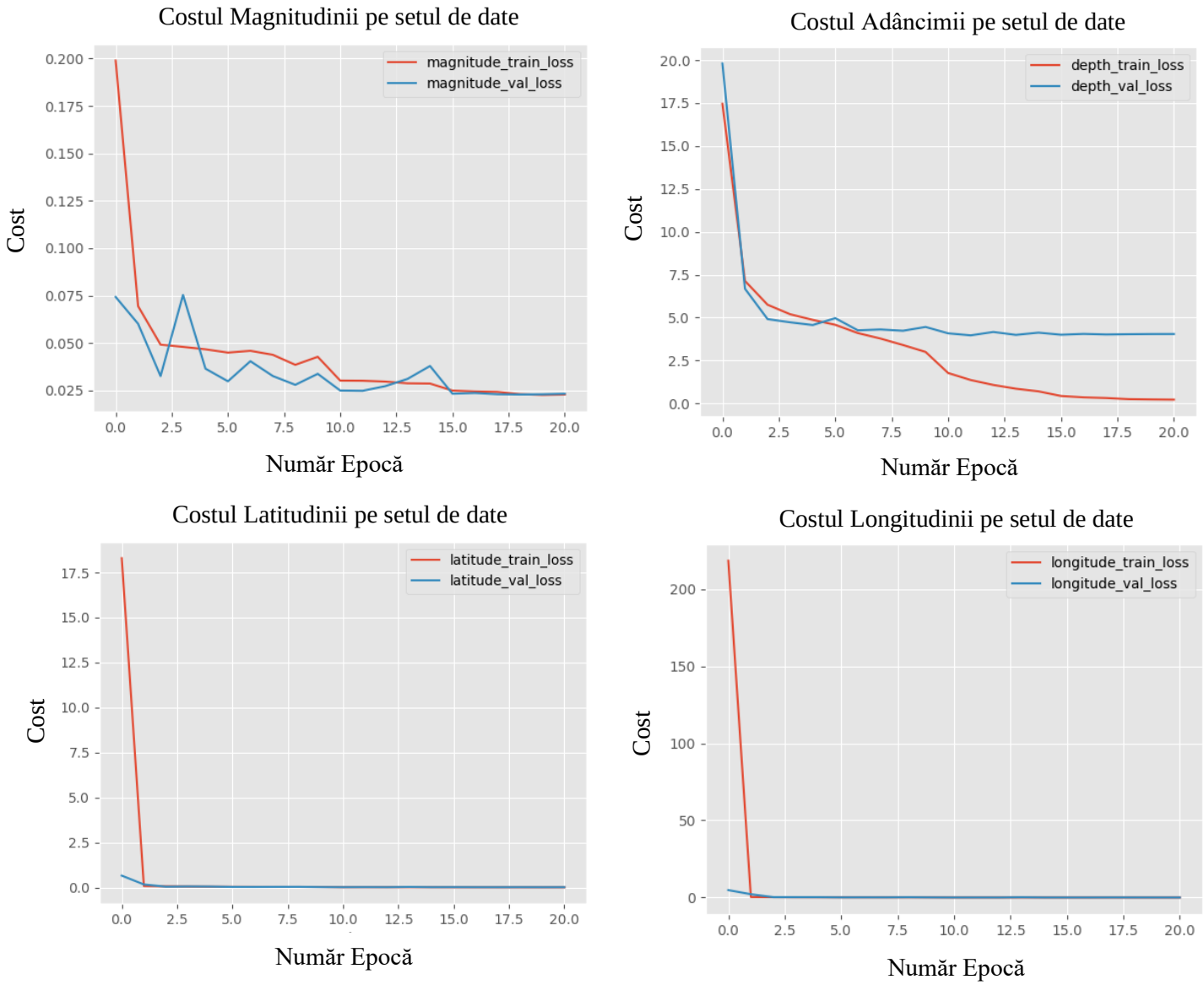


Figura 4.6 Costurile Rețelei Neurale tip ResNet extins

Îmbunătățirea adusă rețelei neurale adânci prin adăugarea noului bloc rezidual nu a dus la o creștere semnificativă a performanțelor, eroare pătratică medie pentru toți parametrii estimați rămânând aproape neschimbată.

Durata unei epoci de antrenare în cazul rețelei ResNet extinsă este în medie de aproximativ 30 de minute, tot procesul de antrenare durând 630 de minute pentru cele 21 de epoci. Procesul de antrenare ajunge să dureze mult mai mult comparativ cu rețeaua tip ResNet prezentată anterior, iar timpul de răspuns pentru o singură dată de intrare crește și el considerabil.

Ținând cont de metoda de regularizare utilizată și de faptul că numărul de epoci după care antrenarea se oprește în cazul în care nu se înregistrează îmbunătățiri pe setul de validare este 8 în această situație, epoca în care s-au obținut parametrii ideali ai rețelei ResNet extinsă este epoca 13.

Valorile funcției de cost pe setul de antrenare și validare scad mult mai rapid și mult mai ferm decât în cazul rețelei de tip ResNet prezentată anterior, dreapta de evoluției a MSE fiind mult mai puțin zgomotoasă. Cu toate acestea, valorile finale pentru adâncime și magnitudine nu se îmbunătățesc suficient de mult comparativ cu rețeaua ResNet anterioară ele fiind cu doar 0,001 în cazul magnitudinii și 0.01 în cazul adâncimii mai bune.

Faptul că erorile pătratice medii scad atât de rapid în cazul latitudinii și longitudinii pe parcursul procesului de antrenare și se stabilizează la fel de rapid ar putea fi cauzat de o situație de supraînvățare din pricina informațiilor insuficiente legate de modul de determinare al epicentrelor, exact cum am menționat în cadrul primei rețele neurale.

Metrică	Antrenare	Validare	Testare
MSE General	0.36517	1.01561	0.98535
MSE Magnitudine	0.03008	0.02477	0.02300
MSE Latitudine	0.01984	0.02272	0.02209
MSE Longitudine	0.04762	0.04880	0.05083
MSE Adâncime	1.36315	3.96617	3.84547

Tabel 4.5 Costurile finale ale rețelei ResNet extinsă

Rezultatele obținute în cadrul procesului de testare arată îmbunătățiri ale performanțelor în cazul magnitudinii, latitudinii și adâncimii, dar o scădere semnificativă în performanță în cazul longitudinii. Primii trei parametrii suferă îmbunătățiri minore ale valorilor coeficientului de determinare și ale abaterilor pătratice medii, iar longitudinea capătă performanțe mai slabe decât cele obținute de către prima implementare a rețelei neurale tip ResNet. De asemenea, abaterea pătratică medie a longitudinii pe setul testare este mai mare decât cea obținută pe setul de validare, iar cea de pe setul de validare este mai mare decât cea pe setul de antrenare. Acest lucru indică clar o supradaptare a rețelei la estimarea longitudinii epicentrului.

Toate aceste rezultate finale obținute pentru rețeaua de tip ResNet extinsă demonstrează că prima implementare sugerată este soluția optimă dintre cele trei și că o creștere a capacității de învățare a rețelei nu va conduce la o extragere mai bună a trăsăturilor și o estimare mai precisă a parametrilor sursei seismice. Costul latenței și al resurselor necesare utilizării rețelei neurale depășesc îmbunătățirile aduse în performanță de către această implementare.

Metrică – set Testare	Magnitudine	Latitudine	Longitudine	Adâncime
Scor R^2	0.9162018	0.7214375	0.36728376	0.8090127

Tabel 4.6 Scorul R^2 pe setul de test - ResNet extins

Graficul de dispersie al valorilor observate cu cele estimate confirmă rezultatele experimentale obținute anterior, afișând o dispersie mai mare și o corelație mai mică a punctelor longitudinii. Magnitudinea, latitudinea și adâncimea afișează cea mai bună corelație a punctelor, în conformitate cu valoarea coeficientului de determinare și cea mai bună abatere pătratică medie a punctelor dintre toate cele trei rețele.

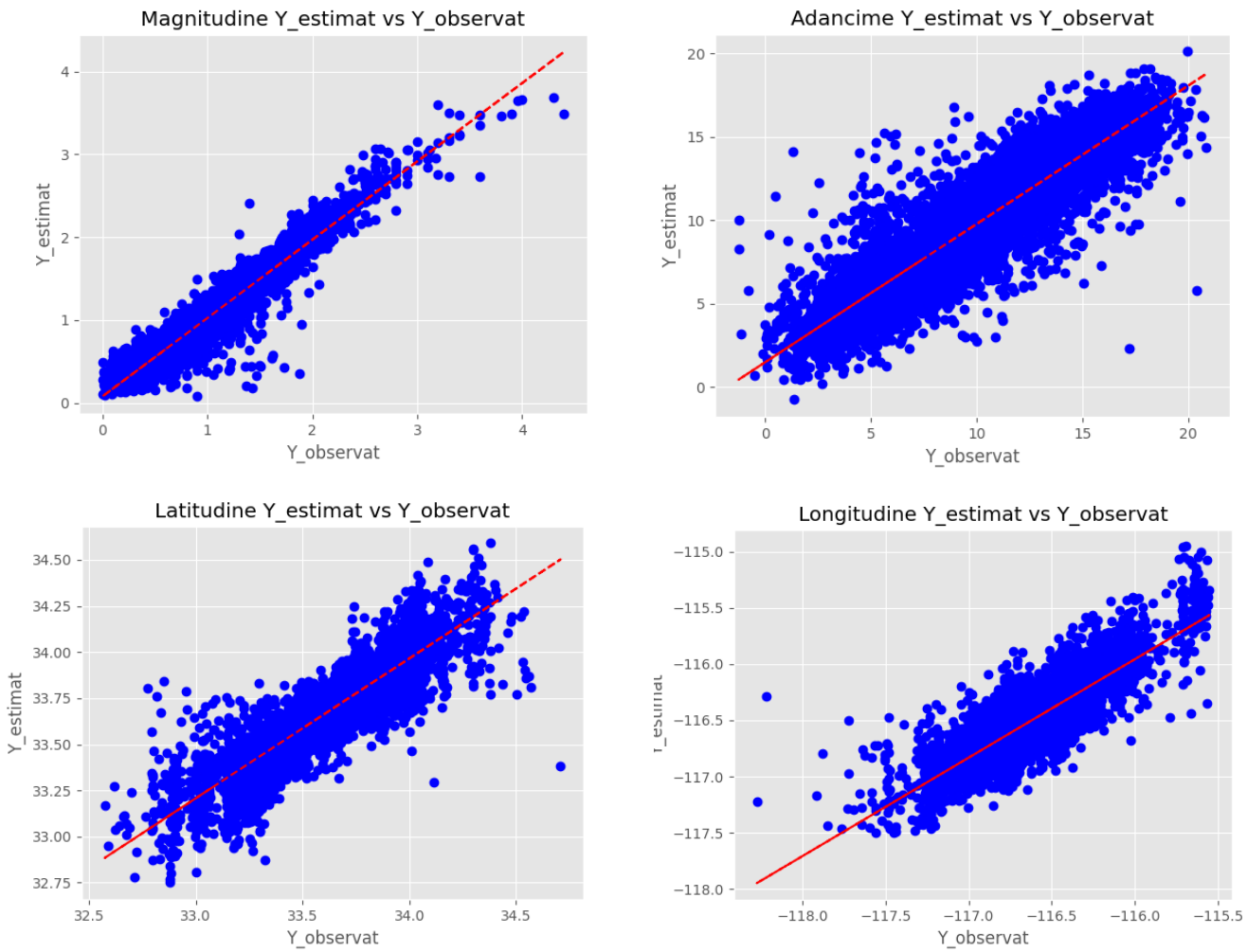


Figura 4.7 Graficul de dispersie al valorilor Y_observat - Y_estimat - ResNet extins

Concluzia finală a acestui experiment este aceea că nu este necesară utilizarea rețelei neurale adânci de tip ResNet extins pentru analiza semnalelor seismice, arhitectura rețelei neurale de tip ResNet propusă inițial fiind suficientă pentru realizarea sarcinilor cu o precizie bună. Implementarea propusă, de ResNet extins, are o latență mai mare, precizie mai slabă în cazul longitudinii și un consum mai ridicat de resurse computaționale pentru executarea algoritmului.

Arhitectura rețelei neurale de tip LeNet deși oferă rezultate comparabile cu cele ale rețelei de tip ResNet, acestea nu sunt suficient de bune pentru utilizarea în cadrul aplicațiilor practice. Atât arhitectura de tip ResNet, cât și cea de tip LeNet nu produc rezultate optime pentru utilizarea lor în cadrul aplicațiilor reale de analiza semnalelor seismice și estimarea parametrilor sursei seismice cu precizie înaltă, însă reprezintă un bun început în studiul acestui domeniu datorită rezultatelor promițătoare oferite mai ales de către ResNet.

4.8. Experiment ResNet – Magnitudine și Epicentru

În urma rezultatelor experimentale obținute pentru arhitectura rețelei neurale de tip ResNet am decis să mai fac un rând de experimente, de această dată fără a mai estima adâncimea sursei seismice. Motivația acestui ultim experiment a fost dată de abaterea pătratică medie foarte mare a adâncimii pe seturile de validare și testare, comparativ cu valorile mici obținute pentru latitudine și longitudine. Aceste rezultate au indicat faptul că estimarea tuturor coordonatelor hipocentrului este o sarcină mult prea complicată de îndeplinit utilizând datele înregistrate de către un singur seismograf.

Exact cum am menționat în Introducere, pentru estimarea coordonatelor unei surse seismice în spațiul tridimensional este nevoie de cel puțin patru senzori care să recepteze semnalul emis de către sursă. Astfel, prin eliminarea sarcinii mult prea dificile de estimare a adâncimii (ce reprezintă coordonata cutremurului în cea de-a treia dimensiune) alături de ceilalți trei parametri, performanța în estimarea magnitudinii, latitudinii și longitudinii ar trebui să crească. Creșterea în performanță a rețelei neurale va fi datorată simplificării sarcinii de estimare a coordonatelor sursei seismice, de la estimarea a trei coordonate aflate într-un plan tridimensional (hipocentru), la estimarea a două coordonate aflate într-un plan bidimensional (epicentrul).

Metrică	Antrenare	Validare	Testare
MSE General	0.01421	0.01475	0.01377
MSE Magnitudine	0.02265	0.0214	0.01957
MSE Latitudine	0.00897	0.00986	0.00956
MSE Longitudine	0.01100	0.01298	0.01218

Tabel 4.7 Costurile finale ale rețelei ResNet - Magnitudine și Epicentru

Numărul total de epoci de antrenare al acestei implementări a fost de 18 epoci, o singură epocă durând tot 20 minute. Tot procesul de antrenare a durat în total 360 de minute, cu doar 40 de minute mai mult decât în cazul estimării tuturor parametrilor sursei seismice.

Ținând cont de metoda de regularizare utilizată și de faptul că numărul de epoci după care antrenarea se oprește în cazul în care nu se înregistrează îmbunătățiri pe setul de validare este 4, epoca în care s-au obținut parametrii ideali ai acestui ResNet este epoca 14.

Eroarea pătratică medie pe setul de validare și testare pentru magnitudine, latitudine și longitudine scade considerabil față de rezultatele obținute în cadrul paragrafului 4.6. Magnitudinea scade de la un MSE pe setul de testare de 0.024 la unul de 0.019, latitudinea de la 0.022 la 0.009, iar longitudinea scade de la 0.033 la 0.012. Îmbunătățirea majoră obținută în estimarea coordonatelor epicentrului confirmă, în acest mod, faptul că estimarea adâncimii alături de latitudine și longitudine scade performanța rețelei neurale când vine vorba de localizarea cutremurului. Acest lucru este datorat corelației ridicate dintre cei trei parametrii, latitudine, longitudine și adâncime.

Valoarea coeficientului de determinare R^2 indică, de asemenea, faptul că în cazul latitudinii și longitudinii, rețeaua neurală adâncă de tip ResNet reușește să coreleze mult mai bine valorile observate cu cele estimate. Modelul rețelei neurale adânci reușește să estimeze cu o precizie mai ridicată epicentrul cutremurelor datorită numărului redus de caracteristici ce trebuie învățate,

comparativ cu cel necesar pentru estimarea hipocentrului. Magnitudinea obține și ea o corelație mult mai bună a valorilor observate cu cele estimate, coeficientul de determinare R^2 ajungând la cea mai mare valoare obținută de până acum, 0.928.

Metrică – set Testare	Magnitudine	Latitudine	Longitudine
Scor R^2	0.9286793	0.8794159	0.84833235

Tabel 4.8 Scorul R^2 pe setul de test - ResNet - Magnitudine și Epicentru

În urma rezultatelor experimentale obținute, se confirmă ipoteza că estimarea adâncimii alături de magnitudine, latitudine și longitudine reduce performanța rețelei neurale adânci din pricina numărului mare de parametrii ce trebuie estimați și, în primul rând, din cauza dificultății ridicate a sarcinii estimării adâncimii hipocentrului. Concluzia finală a experimentului este că o rețea neurală de tip ResNet ce estimează doar magnitudinea și epicentrul oferă cele mai promițătoare rezultate pentru o posibilă aplicație practică de înaltă precizie, abaterile pătratice medii pentru determinarea magnitudinii, latitudinii și longitudinii ajungând la cele mai mici valori dintre toate experimentele efectuate în cadrul acestei lucrări. Cu toate acestea, din punct de vedere practic, erorile pătratice medii continuă să fie insuficient de mici în cazul latitudinii și longitudinii, precizia de estimarea a epicentrelor fiind mai salbă decât rezultatele obținute prin utilizarea metodelor de determinare a coordonatelor utilizând înregistrările mai mult seismometre.

4.9. Vizualizarea estimărilor setului de test

Pentru a vizualiza în mod practic rezultatele estimării parametrilor sursei seismice pe setul de test de către rețeaua neurală adâncă de tip ResNet, am ales reprezentarea epicentrelor reale din setul de date și a epicentrelor estimate de către rețeaua neurală adâncă pe o hartă interactivă.

Harta va fi realizată cu ajutorul bibliotecii open-source „Plotly”, mai exact clasa „graph_objects”. Clasa „graph_objects” pune la dispoziție o metodă pentru definirea aspectului hărții (titlu, stil, zoom, centrare etc.), o metodă de definire a markerelor (etichetelor) ce sunt afișate la ducerea cursorului peste punctele reprezentate pe hartă, o metodă pentru definirea hărții punctelor aferente epicentrelor reale și estimate, o metodă pentru realizarea figurii și o metodă pentru salvarea rezultatului ca fișier interactiv „.html”.[Anexa 2 - 4]

Harta rezultată în final poate fi deschisă de către orice browser web ce suportă Javascript, indicat un browser al unui calculator personal. Harta permite vizualizarea tuturor datelor setului de test, a valorilor reale și estimate de către rețeaua neurală adâncă, în mod interactiv. Markerele punctelor conțin numărul cutremurului în setul de date de test, magnitudine, latitudine, longitudine și adâncimea cutremurului real sau estimat. Stilul hărții este „stamen-terrain”, pentru a permite vizualizarea formelor de relief ale zonei geografice.

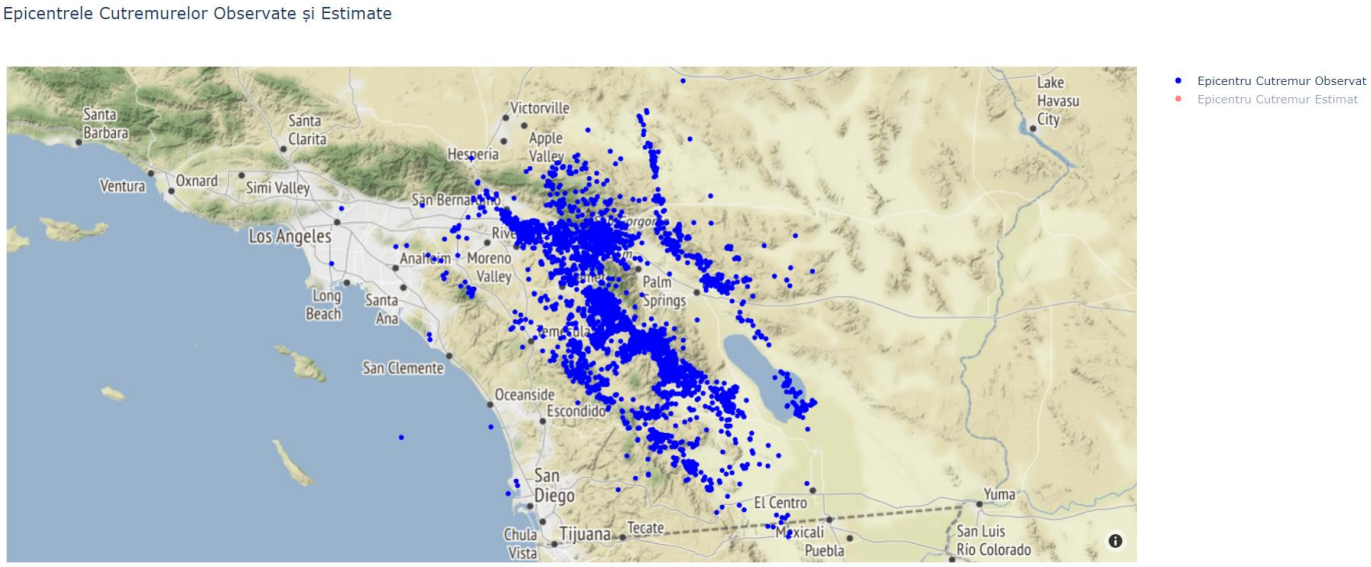


Figura 4.8 Harta cutremurelor observate pentru setul de test

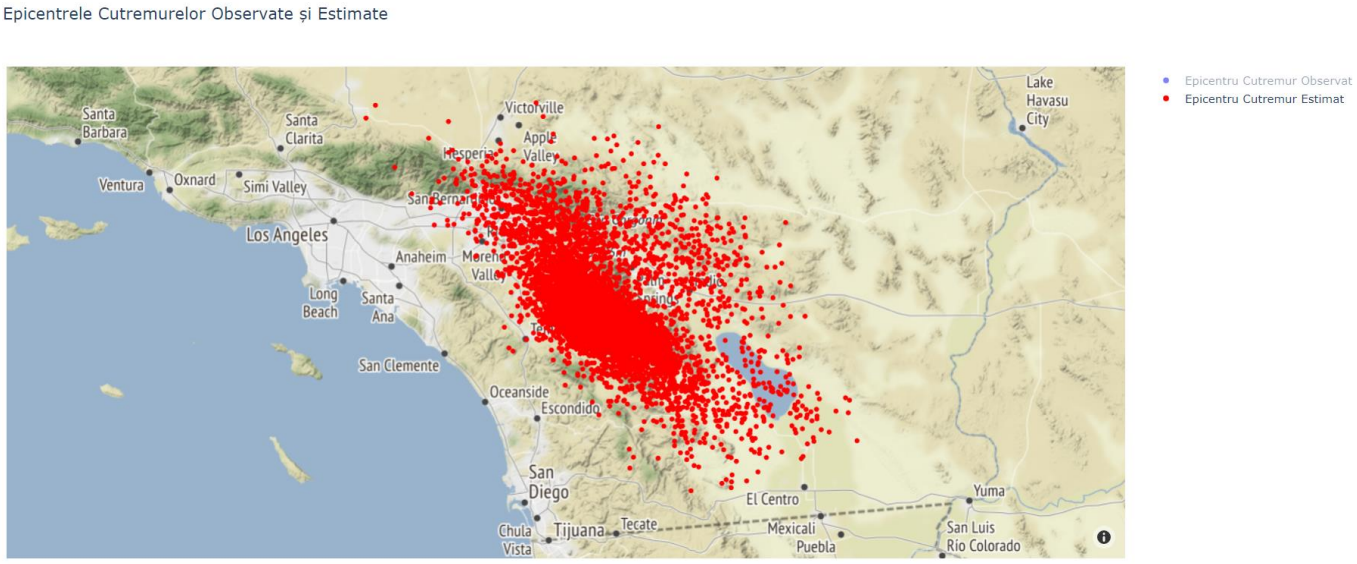


Figura 4.9 Harta cutremurelor estimate pentru setul de test

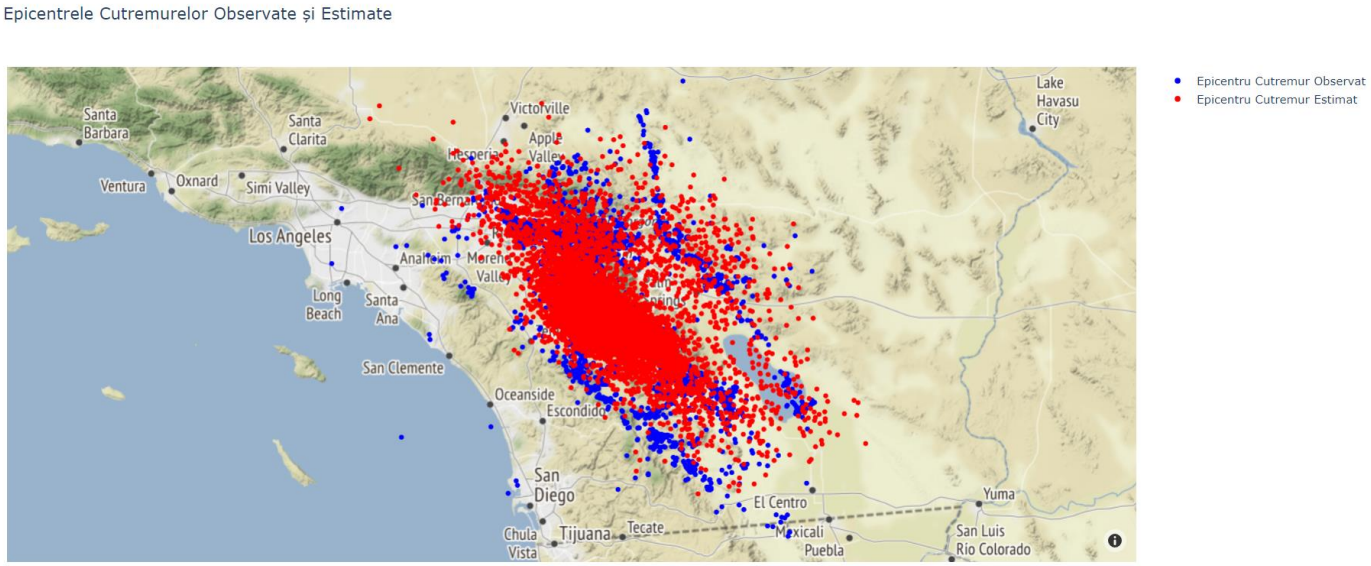


Figura 4.10 Harta cutremurelor observate și estimate pe setul de test - suprapunere

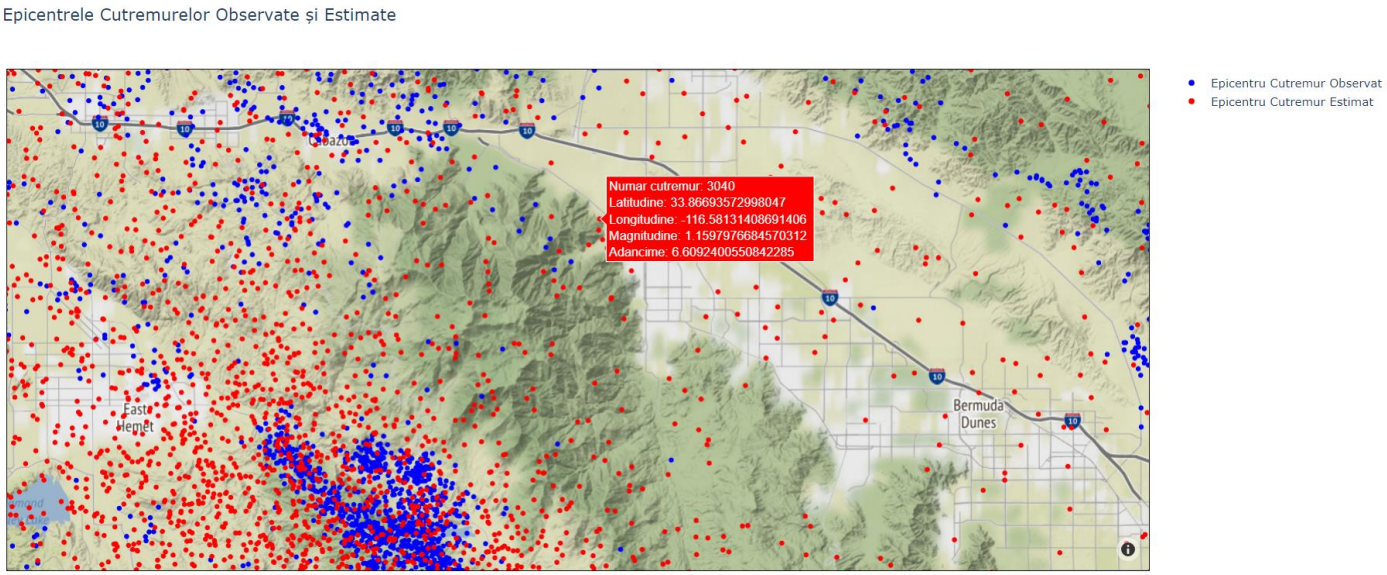


Figura 4.11 Vizualizarea datelor markerelor cutremurelor estimate sau observate

Concluzii

Rezultatele finale ale acestei lucrări au stabilit faptul că arhitectura rețelei neurale adânci sugerată în Tabel 2.1 implementată în stilul arhitecturii rețelelor reziduale, cu salturile de conexiune descrise în Figura 2.3, reprezintă soluția optimă pentru analiza semnalelor seismice și estimarea parametrilor sursei seismice. Atât varianta simplificată a rețelei neurale, arhitectura de tip LeNet insuficient de complexă pentru analiza semnalelor seismice, cât și varianta extinsă a arhitecturii ResNet propuse nu aduc îmbunătățiri suficient de semnificative sau nu oferă performanțe la fel de bune în estimarea anumitor parametrii seismici.

Magnitudinea a reprezentat parametrul seismic cel mai ușor de estimat datorită faptului că și în practică este determinată pe baza analizei amplitudinilor seismogramelor unui singur seismometru. Analiza evoluției în timp a spectrogramelor seismogramelor de manieră empirică permite rețelei neurale adânci extragerea trăsăturilor spectrale definitorii și estimarea magnitudinii pe baza acestora cu o precizie suficient de ridicată.

Cutremurele de mare magnitudine, fiind evenimente care au loc mult mai rar decât activitățile seismice de magnitudini mici (mai mici de 2.5 grade pe scara Richter), vor avea o abatere pătratică medie a valorilor parametrilor estimați mai mare decât în cazul cutremurelor de magnitudine mică, ce au loc mult mai frecvent. Fenomenul este cauzat din pricina lipsei unui număr suficient de mare de eșantioane de cutremure de magnitudine mare ce ar fi putut duce la o generalizare la fel de bună a ambelor tipuri de cutremure în egală măsură.

Latitudinea, longitudine și adâncimea au reprezentat parametrii seismici cei mai greu de estimat din pricina faptului că, în practică, este necesară utilizarea semnalelor seismice receptate de către cel puțin patru senzori (de preferat necoplanari) pentru a putea estima cu exactitate hipocentrul cutremurului în spațiul tridimensional.

Adâncimea rămâne parametrul cel mai greu de estimat atât din pricina poziției sale în cea de a treia dimensiune, cât și din pricina complexității analizei și extragerii trăsăturilor necesare învățării structurii geologice specifice zonei geografice alese.

Longitudinea este mult mai dificil de estimat decât latitudinea din pricina naturii circulare. Față de latitudine, ale cărei valori variază de la -90 la 90 de grade, longitudinea variază de la -180 la 180 de grade și se învârte pe linia internațională a datei. Aceasta înseamnă că o mică diferență de longitudine la ecuator poate reprezenta o distanță mult mai mare pe sol în apropierea polilor. Cu toate acestea, diferența nu este semnificativ de mare în cazul experimentelor realizate.

Eliminarea sarcinii estimării adâncimii din cadrul ieșirilor rețelei neurale adânci sporește performanța în analiza semnalelor seismice și estimarea pe baza caracteristicilor extrase a magnitudinii, latitudinii și longitudinii. Această modificare aduce o scădere considerabilă a MSE pentru latitudine și longitudine, datorită eliminării sarcinii mult prea complexe de a estimat complet hipocentrul cutremurului. Reducere problemei localizării cutremurului de la spațiul tridimensional la cel bidimensional ajută rețeaua foarte mult în concentrarea pe sarcinile ce pot fi realizate cu o precizie mai bună.

Utilizarea rețelelor reziduale în vederea analizei semnalelor seismice și estimării parametrilor surselor de semnal seismic prezintă un potențial foarte mare și o capacitate foarte bună de învățare a extragerii trăsăturilor și estimării ieșirilor pe baza acestora. Deși rezultatele obținute de către rețeaua

neurală adâncă nu sunt suficient apropiate de cele ideale, în cazul latitudinii, longitudinii și adâncimii, experimentele au arătat că rețeaua are posibilitatea de extrage trăsăturile definitorii ale zonei geografice alese din spectrogramele seismogramelor.

Pentru a îmbunătăți performanța estimării epicentrului sau chiar a hipocentrului de către rețeaua neurale adâncă, cea mai sigură modificare ar fi implementarea unei metode complexe de fuzionare a datelor seismice înregistrate de către mai multe seismometre, aflate în aceeași zonă geografică. Altfel, o altă alternativă ar fi implementarea unei arhitecturi de rețea neurală complexă care să utilizeze rezultatul integrala al STFT. Astfel, ar putea fi extrase trăsături și legături în interiorul datelor mult mai avansate decât cele conținute doar de rezultatul real al Transformatei Fourier.

Dacă sugestiile anterioare nu cresc precizia rezultatelor finale ale rețelei neurale adânci propuse suficient de mult, singura opțiune ar fi găsirea unui nou model de rețea neurală adâncă care să poată realiza o analiză mult mai avansată a trăsăturilor spectrogramelor seismogramelor și care să poată estima, folosind aceste informații, ieșirile rețelei cu o toleranță adecvată aplicațiilor practice.

Bibliografie

[1] „Seismology | Earthquakes, Geophysics, & Fault | Britannica”. https://www.britannica.com/science/seismology (data accesării 21 iunie 2023).

[2] N. C. Ristea și A. Radoi, „Complex Neural Networks for Estimating Epicentral Distance, Depth, and Magnitude of Seismic Waves”, *IEEE Geoscience and Remote Sensing Letters*, vol. 19, 2022, doi: 10.1109/LGRS.2021.3059422.

[3] S. M. Mousavi, Y. Sheng, W. Zhu, și G. C. Beroza, „STanford EArthquake Dataset (STEAD): A Global Data Set of Seismic Signals for AI”, *IEEE Access*, vol. 7, pp. 179464–179476, 2019, doi: 10.1109/ACCESS.2019.2947848.

[4] V.-E. Neagoe, „CAPITOLUL Z : INTELIGENŢA COMPUTAŢIONALĂ”, Bucharest, 2023.

[5] „The Concept of Artificial Neurons (Perceptrons) in Neural Networks | by Rukshan Pramoditha | Towards Data Science”. https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc (data accesării 10 iunie 2023).

[6] „Activation Functions in Neural Networks [12 Types & Use Cases]”. https://www.v7labs.com/blog/neural-networks-activation-functions (data accesării 10 iunie 2023).

[7] V.-E. Neagoe, „Capitolul 2 REȚELE NEURALE SUPERVIZATE”, 2023.

[8] „Gentle Introduction to the Adam Optimization Algorithm for Deep Learning - MachineLearningMastery.com”. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (data accesării 18 iunie 2023).

[9] „Various Optimization Algorithms For Training Neural Network | by Sanket Doshi | Towards Data Science”. https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6 (data accesării 10 iunie 2023).

[10] V.-E. Neagoe, „INTRODUCERE IN DEEP LEARNING Modelul Deep Learning (DL)”, 2023.

[11] „7.2. Convolutions for Images — Dive into Deep Learning 1.0.0-beta0 documentation”. https://d2l.ai/chapter_convolutional-neural-networks/conv-layer.html (data accesării 21 iunie 2023).

[12] „Max Pooling Explained | Papers With Code”. https://paperswithcode.com/method/max-pooling (data accesării 19 iunie 2023).

[13] R. M. Stern, „Selected Advanced Topics in Digital Signal Processing”, 2020.

[14] „Various Optimization Algorithms For Training Neural Network | by Sanket Doshi | Towards Data Science”. https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6 (data accesării 10 iunie 2023).

[15] K. Simonyan și A. Zisserman, „Very Deep Convolutional Networks for Large-Scale Image Recognition”, *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, sep. 2014, Data accesării: 10 iunie 2023. [Online]. Disponibil la: https://arxiv.org/abs/1409.1556v6

[16] Y. Lecun, L. Bottou, Y. Bengio, și P. Haffner, „Gradient-based learning applied to document recognition”, *Proceedings of the IEEE*, vol. 86, nr. 11, pp. 2278–2324, 1998, doi: 10.1109/5.726791.

[17] „Batch Normalization | What is Batch Normalization in Deep Learning”. https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/ (data accesării 20 iunie 2023).

[18] „IRIS: SEED Channel Naming”. https://ds.iris.edu/ds/nodes/dmc/data/formats/seed-channel-naming/ (data accesării 11 iunie 2023).

[19] „Types of seismometer – QVSData”. https://qvsdata.com/types-of-seismometer/ (data accesării 11 iunie 2023).

[20] „Mean squared error (MSE) | Definition, Formula, Interpretation, & Facts | Britannica”. https://www.britannica.com/science/mean-squared-error (data accesării 21 iunie 2023).

[21] „Coefficient of Determination (R^2) | Calculation & Interpretation”. https://www.scribbr.com/statistics/coefficient-of-determination/ (data accesării 14 iunie 2023).

Anexa 1
Implementarea modelelor rețelelor neurale adânci

1. Implementare rețelei de tip LeNet:

```
class EqNet(Module):
  def __init__(self, numChannels, outputNodes):
    # apelul constructorului clasei parinte
    super(EqNet,self).__init__()

    # initializare primul strat CONV => RELU => POOL
    self.conv1 = Conv2d(in_channels=numChannels, out_channels=16,
                        kernel_size=(7,7), padding=(3,3))
    self.relu1 = ReLU()
    self.maxpool1 = MaxPool2d(kernel_size=(2,2), stride=(2,2))

    # initializare al doilea strat  CONV => RELU => POOL
    self.conv2 = Conv2d(in_channels=16, out_channels=16, kernel_size=(7,7),
                        padding=(3,3))
    self.relu2 = ReLU()
    self.maxpool2 = MaxPool2d(kernel_size=(2,2), stride=(2,2))

    # initializare primul CONV => RELU => CONV => RELU => POOL = Residual
    self.conv3 = Conv2d(in_channels=16, out_channels=32, kernel_size=(5,5),
                        padding=(2,2))
    self.relu3 = ReLU()
    self.conv4 = Conv2d(in_channels=32, out_channels=32, kernel_size=(5,5),
                        padding=(2,2))
    self.relu4 = ReLU()
    self.maxpool3 = MaxPool2d(kernel_size=(2,2), stride=(2,2))

    # initializare al doilea CONV => RELU => CONV => RELU => POOL = Residual
    self.conv5 = Conv2d(in_channels=32, out_channels=64, kernel_size=(3,3),
                        padding=(1,1))
    self.relu5 = ReLU()
    self.conv6 = Conv2d(in_channels=64, out_channels=64, kernel_size=(3,3),
                        padding=(1,1))
    self.relu6 = ReLU()
    self.maxpool4 = MaxPool2d(kernel_size=(2,2), stride=(2,2))

    # initializare al treilea CONV => RELU => CONV => RELU => POOL= Residual
    self.conv7 = Conv2d(in_channels=64, out_channels=96, kernel_size=(3,3),
                        padding=(1,1))
    self.relu7 = ReLU()
    self.conv8 = Conv2d(in_channels=96, out_channels=96, kernel_size=(3,3),
                        padding=(1,1))
    self.relu8 = ReLU()
    self.maxpool5 = MaxPool2d(kernel_size=(2,2), stride=(2,2))

    # initializare ultimul bloc CONV => RELU => CONV => POOL
    self.conv9 = Conv2d(in_channels=96, out_channels=128, kernel_size=(3,3),
                        padding=(1,1))
    self.relu9 = ReLU()
    self.conv10 = Conv2d(in_channels=128, out_channels=128, kernel_size=(3,3),
                         padding=(1,1))
    self.maxpool6 = MaxPool2d(kernel_size=(2,2), stride=(2,2))
```

```
    # initializare strat ascuns => strat ReLU - fully connected layer
    self.fc1 = Linear(in_features=2048, out_features=1024)
    self.relu10 = ReLU()

    # initializare strat de iesire => Strat Regresie Liniara
    self.fc2 = Linear(in_features=1024, out_features=outputNodes)

  def forward(self, x): # x - lot de date de intrare in retea
    # trecere date prin primul strat CONV -> ReLU -> POOL
    x = self.conv1(x)
    x = self.relu1(x)
    x = self.maxpool1(x)

    # trecere date prin al doilea strat CONV -> ReLU -> POOL
    x = self.conv2(x)
    x = self.relu2(x)
    x = self.maxpool2(x)

    # trecere date prin primul strat CONV => RELU => CONV => RELU => POOL
    x = self.conv3(x)
    x = self.relu3(x)
    x = self.conv4(x)
    x = self.relu4(x)
    x = self.maxpool3(x)

    # trecere date prin al doilea strat CONV => RELU => CONV => RELU => POOL
    x = self.conv5(x)
    x = self.relu5(x)
    x = self.conv6(x)
    x = self.relu6(x)
    x = self.maxpool4(x)

    # trecere date prin al treilea CONV => RELU => CONV => RELU => POOL
    x = self.conv7(x)
    x = self.relu7(x)
    x = self.conv8(x)
    x = self.relu8(x)
    x = self.maxpool5(x)

    # trecere date prin ultimul strat CONV => RELU => CONV => POOL
    x = self.conv9(x)
    x = self.relu9(x)
    x = self.conv10(x)
    x = self.maxpool6(x)

    # redimensionez iesirea stratului anterior si trimit catre stratul ascuns FC
    x = flatten(x,1)
    x = self.fc1(x)
    x = self.relu10(x)

    # aflu iesirea stratului liniar pentru regresia predictiilor
    output = self.fc2(x)

    # intorc rezultatele obtinute
    return output
```

2. Implementarea rețelei de tip ResNet:

```
# definesc blocul rezidual
class block(Module):
  def __init__(self, in_channels, out_channels, kernel_size=(5,5),
               padding=(2,2)):
    super(block, self).__init__()
    self.conv1 = Conv2d(in_channels, out_channels,
                        kernel_size = kernel_size, padding=padding)
    self.bn1 = BatchNorm2d(out_channels)
    self.conv2 = Conv2d(out_channels, out_channels,
                        kernel_size= kernel_size, padding=padding)
    self.bn2 = BatchNorm2d(out_channels)
    self.relu = ReLU(inplace=True)
    self.maxpool = MaxPool2d(kernel_size=(2,2), stride=(2,2))

  def forward(self, x):
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    identity = x
    x = self.conv2(x)
    x = self.bn2(x)
    x += identity

    x = self.relu(x)

    output = self.maxpool(x)

    return output

class ResNet(Module):
  def __init__(self, block, numChannels, outputNodes):
    super(ResNet, self).__init__()
    # initializez primul strat CONV => RELU => POOL
    self.conv1 = Conv2d(in_channels=numChannels, out_channels=16,
                        kernel_size=(7,7), padding=(3,3))
    self.bn1 = BatchNorm2d(16)
    self.relu = ReLU(inplace=True)
    self.maxpool = MaxPool2d(kernel_size=(2,2), stride=(2,2))

    # initializez al doilea strat CONV => RELU => POOL
    self.conv2 = Conv2d(in_channels=16, out_channels=16, kernel_size=(7,7),
                        padding=(3,3))
    self.bn2 = BatchNorm2d(16)

    # straturile de tip ResNet
    self.layer1 = block(in_channels=16, out_channels=32, kernel_size=(5,5),
                        padding=(2,2))
    self.layer2 = block(in_channels=32, out_channels=64, kernel_size=(3,3),
                        padding=(1,1))
    self.layer3 = block(in_channels=64, out_channels=96, kernel_size=(3,3),
                        padding=(1,1))

    # initializez ultimul strat CONV => RELU => CONV => POOL
    self.conv3 = Conv2d(in_channels=96, out_channels=128, kernel_size=(3,3),
                        padding=(1,1))
    self.bn3 = BatchNorm2d(128)

    self.conv4 = Conv2d(in_channels=128, out_channels=128,
                        kernel_size=(3,3), padding=(1,1))
    self.bn4 = BatchNorm2d(128)
```

```
    # initializez stratul FC => ReLU layers - fully connected
    self.fc1 = Linear(in_features=2048, out_features=1024)

    # initializez stratul FC => Linear *Regression*
    self.fc2 = Linear(in_features=1024, out_features=outputNodes)

  def forward(self, x):
  # trimit intrarea prin primul bloc CONV -> ReLU -> POOL
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    # trimit rezultatele prin al doilea bloc CONV -> ReLU -> POOL
    x = self.conv2(x)
    x = self.bn2(x)
    x = self.relu(x)
    x = self.maxpool(x)

    # trimit datele prin blocurile reziduale
    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)

    # trimit datele prin blocul CONV => RELU => CONV => POOL
    x = self.conv3(x)
    x = self.bn3(x)
    x = self.relu(x)
    x = self.conv4(x)
    x = self.bn4(x)
    x = self.maxpool(x)

    # redimensionez iesirea stratului anterior si trimit catre stratul ascuns FC
    x = flatten(x, 1)
    x = self.fc1(x)
    x = self.relu(x)

    # trimit datele catre stratul Linear pentru realizarea regresiei
    output = self.fc2(x)

    # intorc rezultatele prezise
    return output
```

3. Implementarea rețelei de tip ResNet cu bloc rezidual adițional:

```
class STEADDataset(Dataset):

# -----------------------

  def getSpectrogram(self, waveforms):
    # defining axis
    EW = waveforms[:,0]
    NS = waveforms[:,1]
    Vert = waveforms[:,2]

    EW_ft = lib.stft(EW,n_fft=1024, hop_length=16)
    NS_ft = lib.stft(NS,n_fft=1024, hop_length=16)
    Vert_ft = lib.stft(Vert,n_fft=1024, hop_length=16)

    EW_db = lib.amplitude_to_db(np.abs(EW_ft), ref=np.max)
    NS_db = lib.amplitude_to_db(np.abs(NS_ft), ref=np.max)
    Vert_db = lib.amplitude_to_db(np.abs(Vert_ft), ref=np.max)

    spectrograms = np.array([EW_db, NS_db, Vert_db])

    return spectrograms


class EqNet(Module):
  def __init__(self, block, numChannels, outputNodes):
    super(EqNet, self).__init__()
    # initializez primul strat CONV => RELU => POOL
    self.conv1 = Conv2d(in_channels=numChannels, out_channels=16,
                        kernel_size=(7,7), padding=(3,3))
    self.bn1 = BatchNorm2d(16)
    self.relu = ReLU(inplace=True)
    self.maxpool = MaxPool2d(kernel_size=(2,2), stride=(2,2))

    # initializez al doilea strat CONV => RELU => POOL
    self.conv2 = Conv2d(in_channels=16, out_channels=16, kernel_size=(7,7),
                        padding=(3,3))
    self.bn2 = BatchNorm2d(16)

    # intializez blocurile reziduale
    self.layer1 = block(in_channels=16, out_channels=32, kernel_size=(5,5),
                        padding=(2,2))
    self.layer2 = block(in_channels=32, out_channels=64, kernel_size=(5,5),
                        padding=(2,2))
    self.layer3 = block(in_channels=64, out_channels=96, kernel_size=(3,3),
                        padding=(1,1))
    self.layer4 = block(in_channels=96, out_channels=128, kernel_size=(3,3),
                        padding=(1,1))

    # initializez ultimul strat CONV => RELU => CONV => POOL
    self.conv3 = Conv2d(in_channels=128, out_channels=256,
                        kernel_size=(3,3), padding=(1,1))
    self.bn3 = BatchNorm2d(256)
    self.conv4 = Conv2d(in_channels=256, out_channels=256,
                        kernel_size=(3,3), padding=(1,1))
    self.bn4 = BatchNorm2d(256)

    # initializez primul strat FC => ReLU
    self.fc1 = Linear(in_features=2048, out_features=1024)

    # initializez al doilea strat FC => (Regresie) Liniara
    self.fc2 = Linear(in_features=1024, out_features=outputNodes)
```

```
def forward(self, x):
# trimit datele de intrare prin primul bloc of CONV -> ReLU -> POOL
  x = self.conv1(x)
  x = self.bn1(x)
  x = self.relu(x)
  x = self.maxpool(x)

  # trimit datele prin al doilea bloc CONV -> ReLU -> POOL
  x = self.conv2(x)
  x = self.bn2(x)
  x = self.relu(x)
  x = self.maxpool(x)

  # trimit datele prin blocurile reziduale
  x = self.layer1(x)
  x = self.layer2(x)
  x = self.layer3(x)
  x = self.layer4(x)

  # trimit datele prin ultimul bloc CONV => RELU => CONV => POOL
  x = self.conv3(x)
  x = self.bn3(x)
  x = self.relu(x)
  x = self.conv4(x)
  x = self.bn4(x)
  x = self.maxpool(x)

  # redimensionez datele si le trimit primului strat FC
  x = flatten(x, 1)
  x = self.fc1(x)
  x = self.relu(x)

  # trimit datele catre ultimul strat FC si realizez regresia liniara
  output = self.fc2(x)

  # intorc estimarile datelor de iesire
  return output
```

Anexa 2
Unelte Software utilizate în procesul de antrenare și testare

1. Dataloader Personalizat:

```
class STEADDataset(Dataset):

  def __init__(self,csv_file,hdf5_file,transform=None):
    """
      Args:
        csv_file (string): calea catre fisierul csv cu etichete.
        hdf5_file (string): fisierul cu formele de unda.
        transform (callable, optional): transformare a unui esantion.
    """
    self.tags = pd.read_csv(csv_file)
    self.hdf5_file = hdf5_file
    self.traces = self.tags['trace_name'].to_list()
    self.transform = transform

  def __len__(self):
    return len(self.traces)

  def __getitem__(self,idx):
    if torch.is_tensor(idx):
      idx = idx.tolist()

    dataset = h5py.File(self.hdf5_file,'r')
    tracename = self.traces[idx]
    waveform = dataset.get('data/'+tracename)
    data = np.array(waveform)
    spectrograms = self.getSpectrogram(data)

    sample = {'spectrograms':spectrograms,
              'source_magnitude':waveform.attrs['source_magnitude'],
              'source_latitude':waveform.attrs['source_latitude'],
              'source_longitude':waveform.attrs['source_longitude'],
              'source_depth_km':waveform.attrs['source_depth_km']}

    if self.transform:
      sample = self.transform(sample)

    dataset.close()
    return sample

  def getSpectrogram(self, waveforms):
    # defining axis
    EW = waveforms[:,0]
    NS = waveforms[:,1]
    Vert = waveforms[:,2]

    EW_ft = lib.stft(EW,n_fft=1024, hop_length=16)
    NS_ft = lib.stft(NS,n_fft=1024, hop_length=16)
    Vert_ft = lib.stft(Vert,n_fft=1024, hop_length=16)

    EW_db = lib.amplitude_to_db(np.abs(EW_ft), ref=np.max)
    NS_db = lib.amplitude_to_db(np.abs(NS_ft), ref=np.max)
    Vert_db = lib.amplitude_to_db(np.abs(Vert_ft), ref=np.max)
```

```
    spectrograms = np.array([EW_db, NS_db, Vert_db])

    return spectrograms

class ToTensor(object):
    """Converteste NumPY ndarrays la Tensori."""

    def __call__(self, sample):
      if sample['source_depth_km'] == "None":
        return None
      else:
        spectrograms = sample['spectrograms']
        results = np.array([sample['source_magnitude'],
                            sample['source_latitude'],
                            sample['source_longitude'],
                            sample['source_depth_km']], dtype=np.float32)

        return {'spectrograms': torch.from_numpy(spectrograms),
                'results':  torch.from_numpy(results)}
```

2. Early Stopper:

```
# definesc clasa EarlyStopping
class EarlyStopping():
  def __init__(self, patience = 1, min_delta = 0):
    self.patience = patience
    self.min_delta = min_delta
    self.counter = 0
    self.min_validation_loss = np.inf
    self.best_epoch = 0
    self.best_train = [None] * 5
    self.best_val = [None] * 5

  def earlyStop(self, validation_loss, epoch, TrainLoss, ValLoss):
    if validation_loss <= self.min_validation_loss:
      print("[INFO] In EPOCH {} the loss value improved from {:.5f} to
{:.5f}".format(epoch, self.min_validation_loss, validation_loss))
      self.min_validation_loss = validation_loss
      self.counter = 0
      self.best_epoch = epoch
# salvez ponderile celui mai performant model de pana acum
      torch.save(model.state_dict(), f"{models_dir}/ResNet_state_dict.pt")
      self.setBestLosses(TrainLoss, ValLoss)

    elif validation_loss > (self.min_validation_loss + self.min_delta):
      self.counter += 1
      print("[INFO] In EPOCH {} the loss value did not improve from {:.5f}. This
is the {} EPOCH in a row.".format(epoch, self.min_validation_loss,
                                  self.counter))
      if self.counter >= self.patience:
        return True
    return False

  def setCounter(self, counter_state):
    self.counter = counter_state

  def setMinValLoss(self, ValLoss):
    self.min_validation_loss = ValLoss

  def setBestLosses(self, TrainLoss, ValLoss):
    self.best_train = TrainLoss
    self.best_val = ValLoss
```

```
  def setBestEpoch(self, bestEpoch):
    self.best_epoch = bestEpoch

  def getBestTrainLosses(self):
    return self.best_train

  def getBestValLosses(self):
    return self.best_val

  def getBestEpoch(self):
    return self.best_epoch

  def saveLossesLocally(self):
    np.save(f'{models_dir}/losses_train.npy', np.array(self.best_train))
    np.save(f'{models_dir}/losses_val.npy', np.array(self.best_val))

  def loadLossesLocally(self):
    self.best_train = np.load(f'{models_dir}/losses_train.npy')
    self.best_val = np.load(f'{models_dir}/losses_val.npy')
```

3. Bucla de antrenare și afișarea rezultatelor (ResNet):

```
for e in range(last_epoch, EPOCHS):
  # setez modelul in modul antrenare
  model.train()

  # init. variabilele pentru costurile totale pe antrenare si validare
  magnitudeTrainLoss = 0
  latitudeTrainLoss = 0
  longitudeTrainLoss = 0
  depthTrainLoss = 0
  generalTrainLoss = 0

  magnitudeValLoss = 0
  latitudeValLoss = 0
  longitudeValLoss = 0
  depthValLoss = 0
  generalValLoss = 0

  for sampled_batch in trainDataLoader:
    # verific daca lotul nu este nul
    if sampled_batch is None:
      continue

    # send the input to the device
    (x, y) = (sampled_batch['spectrograms'].to(device),
              sampled_batch['results'].to(device))

    # calculez estimarile si costurile
    pred = model(x)
    loss = lossFn(pred, y)
    # resetez gradientii, fac backprop, actualizez ponderile
    opt.zero_grad()
    loss.backward()
    opt.step()

    # adun costurile acumulate până acum
    generalTrainLoss += loss.cpu().detach().numpy()
    magnitudeTrainLoss += lossFn(pred[:,0],y[:,0]).cpu().detach().numpy()
    latitudeTrainLoss += lossFn(pred[:,1],y[:,1]).cpu().detach().numpy()
    longitudeTrainLoss += lossFn(pred[:,2],y[:,2]).cpu().detach().numpy()
```

```
    depthTrainLoss += lossFn(pred[:,3],y[:,3]).cpu().detach().numpy()

  # opresc autograd pentru validare
  with torch.no_grad():
    # setez modelul in modul evaluare
    model.eval()

    # bucla for peste setul de validare
    for sampled_batch in valDataLoader:
  # verific daca lotul nu este nul
      if sampled_batch is None:
        continue

      # trimit datele de intrare la dispozitiv
      (x, y) = (sampled_batch['spectrograms'].to(device),
                sampled_batch['results'].to(device))

      # fac estimarile si calculez costurile
      pred = model(x)
      generalValLoss += lossFn(pred, y).cpu().detach().numpy()
      magnitudeValLoss += lossFn(pred[:,0],y[:,0]).cpu().detach().numpy()
      latitudeValLoss += lossFn(pred[:,1],y[:,1]).cpu().detach().numpy()
      longitudeValLoss += lossFn(pred[:,2],y[:,2]).cpu().detach().numpy()
      depthValLoss += lossFn(pred[:,3],y[:,3]).cpu().detach().numpy()

  # calculez costirle medii pe lotul de antrenare si validare
  avgGeneralTrainLoss = generalTrainLoss / trainSteps
  avgMagnitudeTrainLoss =  magnitudeTrainLoss / trainSteps
  avgLatitudeTrainLoss =  latitudeTrainLoss / trainSteps
  avgLongitudeTrainLoss =  longitudeTrainLoss / trainSteps
  avgDepthTrainLoss =  depthTrainLoss / trainSteps

  avgGeneralValLoss = generalValLoss / valSteps
  avgMagnitudeValLoss =  magnitudeValLoss / valSteps
  avgLatitudeValLoss =  latitudeValLoss / valSteps
  avgLongitudeValLoss =  longitudeValLoss / valSteps
  avgDepthValLoss =  depthValLoss / valSteps

  # adaug inca un pas printr-o epoca in LR Scheduler
  scheduler.step(avgGeneralValLoss)

  # actualizez istoricul costurile
  H["magnitude_train_loss"].append(avgMagnitudeTrainLoss)
  H["latitude_train_loss"].append(avgLatitudeTrainLoss)
  H["longitude_train_loss"].append(avgLongitudeTrainLoss)
  H["depth_train_loss"].append(avgDepthTrainLoss)

  H["magnitude_val_loss"].append(avgMagnitudeValLoss)
  H["latitude_val_loss"].append(avgLatitudeValLoss)
  H["longitude_val_loss"].append(avgLongitudeValLoss)
  H["depth_val_loss"].append(avgDepthValLoss)

  # salvez dictionarele de stare - checkpoint
  torch.save({
            'epoch': e,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': opt.state_dict(),
            'scheduler_state_dict': scheduler.state_dict(),
            'train_loss_history': H
            }, f"{models_dir}/train_state_dict_final.pt")

  # afisez informatiile modelului pentru epoca de antrenare
  print("[INFO] EPOCH: {}/{} ...".format(e+1, EPOCHS))
```

```
  print("Train loss (General, Magnitude, Latitude, Longitude, Depth): {:.5f},
{:.5f} , {:.5f} , {:.5f}, {:.5f}".format(
      avgGeneralTrainLoss, avgMagnitudeTrainLoss, avgLatitudeTrainLoss,
      avgLongitudeTrainLoss, avgDepthTrainLoss))
  print("Val loss (General, Magnitude, Latitude, Longitude, Depth): {:.5f},
{:.5f} , {:.5f} , {:.5f}, {:.5f}".format(
      avgGeneralValLoss, avgMagnitudeValLoss, avgLatitudeValLoss,
      avgLongitudeValLoss, avgDepthValLoss))

  # verific daca eroarea pe setul de validare e cea mai buna
  if early_stopper.earlyStop(avgGeneralValLoss, (e+1),
[avgGeneralTrainLoss, avgMagnitudeTrainLoss, avgLatitudeTrainLoss,
avgLongitudeTrainLoss, avgDepthTrainLoss],
[avgGeneralValLoss, avgMagnitudeValLoss, avgLatitudeValLoss,
avgLongitudeValLoss, avgDepthValLoss]):
    # daca se depaseste asteptarea, opresc antrenarea
    print("[INFO] Early Stopping the train process. The patience has been
exceeded!")
    print("================================================================")
    break

  print("================================================================")

# fac masuratorile finale dupa finalizarea procesului de antrenare
endTime = time.time()
print("[INFO] Total time taken to train the model: {:.2f}s".format(endTime-
startTime))
# salvez local costurile cele mai bune si epoca cu valorile optime
print("[INFO] The best loss value was found in EPOCH {} where the performance
was {:.5f}. Model's parameters saved!".format(early_stopper.getBestEpoch(),
early_stopper.getBestValLosses()[0]))
early_stopper.saveLossesLocally()

# realizez graficele costului pe procesul de antrenare
plt.style.use("ggplot")

# realizez graficul costului magnitudinii pe seturile de antrenare si validare
plt.figure("magnitude_loss").clear()
plt.plot(H["magnitude_train_loss"], label="magnitude_train_loss",
linestyle="solid")
plt.plot(H["magnitude_val_loss"], label="magnitude_val_loss", linestyle="solid")
plt.title("Costul Magnitudinii pe setul de date")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="upper right")
plt.savefig(f"{plots_dir}/ResNet3_magnitude_loss.png")

# realizez graficul costului latitudinii pe seturile de antrenare si validare
plt.figure("latitude_loss").clear()
plt.plot(H["latitude_train_loss"], label="latitude_train_loss",
linestyle="solid")
plt.plot(H["latitude_val_loss"], label="latitude_val_loss", linestyle="solid")
plt.title("Costul Latitudinii pe setul de date")
plt.xlabel("Numar epoca")
plt.ylabel("Cost")
plt.legend(loc="upper right")
plt.savefig(f"{plots_dir}/ResNet3_latitude_loss.png")

# realizez graficul costului longitudinii pe seturile de antrenare si validare
plt.figure("longitude_loss").clear()
plt.plot(H["longitude_train_loss"], label="longitude_train_loss",
linestyle="solid")
plt.plot(H["longitude_val_loss"], label="longitude_val_loss", linestyle="solid")
```

```
plt.title("Costul Longitudinii pe setul de date")
plt.xlabel("Numar epoca")
plt.ylabel("Cost")
plt.legend(loc="upper right")
plt.savefig(f"{plots_dir}/ResNet3_longitude_loss.png")

# realizez graficul costului adancimii pe seturile de antrenare si validare
plt.figure("depth_loss").clear()
plt.plot(H["depth_train_loss"], label="depth_train_loss", linestyle="solid")
plt.plot(H["depth_val_loss"], label="depth_val_loss", linestyle="solid")
plt.title("Costul Adancimii pe setul de date")
plt.xlabel("Numar epoca")
plt.ylabel("Cost")
plt.legend(loc="upper right")
plt.savefig(f"{plots_dir}/ResNet3_depth_loss.png")
```

4. Bucla de testare și afișarea rezultatelor (ResNet):

```
# opresc autograd pentru testarea retelei
with torch.no_grad():
  # setez modeleul in modul evaluare
  model.eval()

  # initializez costurile totale pe setul de test
  generalTestLoss = 0
  magnitudeTestLoss = 0
  latitudeTestLoss =  0
  longitudeTestLoss = 0
  depthTestLoss = 0

  # interez prin setul de test
  for sampled_batch in testDataLoader:
    if sampled_batch is None:
      continue

    # trimit datele de intrare catre dispozitiv
    (x, y) =
(sampled_batch['spectrograms'].to(device),sampled_batch['results'].to(device))
    test_results["true_values"].append(y.cpu().detach().numpy().tolist())

    # fac estimarile si le adaug in lista
    pred = model(x)
    test_results["pred_values"].append(pred.cpu().detach().numpy().tolist())

    generalTestLoss += lossFn(pred, y).cpu().detach().numpy()
    magnitudeTestLoss += lossFn(pred[:,0], y[:,0]).cpu().detach().numpy()
    latitudeTestLoss += lossFn(pred[:,1], y[:,1]).cpu().detach().numpy()
    longitudeTestLoss += lossFn(pred[:,2], y[:,2]).cpu().detach().numpy()
    depthTestLoss += lossFn(pred[:,3], y[:,3]).cpu().detach().numpy()
    r2score_metric.update(pred, y)

  # generez MSE si scorul R2 pe setul de test
  avgGeneralTestLoss = generalTestLoss / testSteps
  avgMagnitudeTestLoss = magnitudeTestLoss / testSteps
  avgLatitudeTestLoss =  latitudeTestLoss / testSteps
  avgLongitudeTestLoss = longitudeTestLoss / testSteps
  avgDepthTestLoss =  depthTestLoss / testSteps

  r2score_value = r2score_metric.compute()
```

```
  print("[INFO] Loss/Accuracy values obtained on the test set")
  print("[INFO] Test loss (General, Magnitude, Latitude, Longitude, Depth):
{:.5f}, {:.5f} , {:.5f} , {:.5f}, {:.5f}".format(
      avgGeneralTestLoss, avgMagnitudeTestLoss, avgLatitudeTestLoss,
avgLongitudeTestLoss, avgDepthTestLoss))
  print("[INFO] R2 Score obtained on the test set:
{}".format(r2score_value.cpu().detach().numpy()))

# realizez și salvez graficele de dispersie pentru Y_observat - Y_estimat
plt.style.use("ggplot")

test_true = []
test_pred = []

for i in range(len(test_results["true_values"])):
  test_true.extend(test_results["true_values"][i])
  test_pred.extend(test_results["pred_values"][i])

test_true = np.array(test_true)
test_pred = np.array(test_pred)

# realizez graficul dispersiei punctelor de magnitudine Y_observat - Y_estimat
plt.figure("magnitude_true-pred").clear()
plt.plot(test_true[:,0], test_pred[:,0], "ob")
m, b = np.polyfit(test_true[:,0], test_pred[:,0], 1)
plt.plot(test_true[:,0], m*test_true[:,0]+b,"--r")
plt.title("Magnitudine Y_estimat vs Y_observat")
plt.xlabel("Y_observat")
plt.ylabel("Y_estimat")
plt.savefig(f"{plots_dir}/ResNet3_magnitude_true-pred.png")

# realizez graficul dispersiei punctelor de latitudine Y_observat - Y_estimat
plt.figure("latitude_true-pred").clear()
plt.plot(test_true[:,1], test_pred[:,1], "ob")
m, b = np.polyfit(test_true[:,1], test_pred[:,1], 1)
plt.plot(test_true[:,1], m*test_true[:,1]+b,"--r")
plt.title("Latitudine Y_estimat vs Y_observat")
plt.xlabel("Y_observat")
plt.ylabel("Y_estimat")
plt.savefig(f"{plots_dir}/ResNet3_latitude_true-pred.png")

# realizez graficul dispersiei punctelor de longitudine Y_observat - Y_estimat
plt.figure("longitude_true-pred").clear()
plt.plot(test_true[:,2], test_pred[:,2], "ob")
m, b = np.polyfit(test_true[:,2], test_pred[:,2], 1)
plt.plot(test_true[:,2], m*test_true[:,2]+b,"--r")
plt.title("Longitudine Y_estimat vs Y_observat")
plt.xlabel("Y_observat")
plt.ylabel("Y_estimat")
plt.savefig(f"{plots_dir}/ResNet3_longitude_true-pred.png")

# realizez graficul dispersiei punctelor de adancime Y_observat - Y_estimat
plt.figure("depth_true-pred").clear()
plt.plot(test_true[:,3], test_pred[:,3], "ob")
m, b = np.polyfit(test_true[:,3], test_pred[:,3], 1)
plt.plot(test_true[:,3], m*test_true[:,3]+b,"--r")
plt.title("Adancime Y_estimat vs Y_observat")
plt.xlabel("Y_observat")
plt.ylabel("Y_estimat")
plt.savefig(f"{plots_dir}/ResNet3_depth_true-pred.png")

import plotly.graph_objects as go
```

```
# definesc datele pentru valorile reale (gt) si cele estimate (pred)
gt_magnitudes = test_true[:,0]
gt_latitudes = test_true[:,1]
gt_longitudes = test_true[:,2]
gt_depth = test_true[:,3]

pred_magnitudes = test_pred[:,0]
pred_latitudes = test_pred[:,1]
pred_longitudes = test_pred[:,2]
pred_depth = test_pred[:,3]

# definesc aspectul hartii
layout = go.Layout(
    mapbox=dict(
        center=dict(lat=33.5, lon=-116.8),
        zoom=8,
        style='stamen-terrain'
    ),
    title='Epicentrele cutremurelor observate și estimate')

# definesc textul afisat la vizualizare punctelor
gt_hover_text = ['Numar cutremur: {} <br>Latitudine: {}<br>Longitudine: {}
<br>Magnitudine: {} <br>Adancime: {}'.format(num, lat, lon, magn, depth)
                 for num, (lat, lon, magn, depth) in enumerate(zip(gt_latitudes,
gt_longitudes, gt_magnitudes, gt_depth))]
pred_hover_text = ['Numar cutremur: {} <br>Latitudine: {}<br>Longitudine: {}
<br>Magnitudine: {} <br>Adancime: {}'.format(num, lat, lon, magn, depth)
                   for num, (lat, lon, magn, depth) in
enumerate(zip(pred_latitudes, pred_longitudes, pred_magnitudes, pred_depth))]

# creez cutiile hartii de dispersie a coord. epicentrelor reale si estim.
gt_trace = go.Scattermapbox(
    lat=gt_latitudes,
    lon=gt_longitudes,
    mode='markers',
    marker=dict(5
        color='blue'
    ),
    name='Date seismice observate',
    hovertext=gt_hover_text,
    hoverinfo='text'
)

pred_trace = go.Scattermapbox(
    lat=pred_latitudes,
    lon=pred_longitudes,
    mode='markers',
    marker=dict(
        color='red'
    ),
    name='Date seismice estimate',
    hovertext=pred_hover_text,
    hoverinfo='text'
)

# creez figura, adaug punctele si aspectul
fig = go.Figure(data=[gt_trace, pred_trace], layout=layout)

# salvez harta ca fisier html
fig.write_html(f"{maps_dir}/ResNet3_map.html")
```

Anexa 3
Inițializări și importuri

1. Conectare și atașare Google Drive la sesiune:

```
drive.mount('/content/drive')
plots_dir = '/content/drive/My Drive/Plots/full'
models_dir = '/content/drive/My Drive/Models/full'
maps_dir = '/content/drive/My Drive/Map'
```

2. Redefinire funcție collate (suprapunere):

```
def custom_collate_fn(batch):
  # filtrez estanioanele None (regasite in cazul etichetei adancimii)
  filtered_batch = [sample for sample in batch if sample is not None]
  if len(filtered_batch) == 0:
    # daca lotul nu are niciun esantion - toate None
    return None
  else:
    # creez noul batch cu esantioanele None eliminate
    return default_collate(filtered_batch)
```

3. Inițializare rată de învățare, dimensiune lot, număr epoci și împărțirea setului de date:

```
# definesc hiperparametrii pentru antrenare
INIT_LR = 1*1e-4
BATCH_SIZE = 32
EPOCHS = 50

# defines impartirea intre antrenare si validare
TRAIN_SPLIT = 0.70
VAL_TEST_SPLIT = 0.15

# setez dispozitivul ce va fi utilizat pentru antrenare
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("[INFO] device used for training...{}".format(device))

# definesc impartirea datelor antrenare/validare/testare
print("[INFO] generating the train/validation split...")
numTrainSamples = int(len(STEAD_dataset)*TRAIN_SPLIT)
numValSamples = int(len(STEAD_dataset)*VAL_TEST_SPLIT)
numTestSamples = int(len(STEAD_dataset)-(numTrainSamples+numValSamples))

(trainData, valData, testData) = random_split(STEAD_dataset,[numTrainSamples,
numValSamples, numTestSamples], generator=torch.Generator().manual_seed(19))

# initializez Dataloader-ele pentru cele trei seturi
trainDataLoader = DataLoader(trainData, shuffle=True, batch_size=BATCH_SIZE,
collate_fn = custom_collate_fn)
valDataLoader = DataLoader(valData, batch_size=BATCH_SIZE, collate_fn =
custom_collate_fn)
testDataLoader = DataLoader(testData, batch_size=BATCH_SIZE, collate_fn =
custom_collate_fn)
```

```
# calculez numărul de pasi dintr-o epoca pentru fiecare set
trainSteps = len(trainDataLoader.dataset) // BATCH_SIZE
valSteps = len(valDataLoader.dataset) // BATCH_SIZE
testSteps = len(testDataLoader.dataset) // BATCH_SIZE
```

4. Inițializările efectuate pentru procesul de antrenare și validare:

```
# incarc starile precedente din Google Drive
# previous_state = torch.load(f"{models_dir}/train_state_dict_final.pt")

# initializez modelul ResNet
print("[INFO] initializing the ResNet model...")
model = EqNet(block=block, numChannels=3, outputNodes=4).to(device)
# model.load_state_dict(previous_state['model_state_dict'])

# initializez functia de optimizare si programatorul ratei de invatare
opt = Adam(model.parameters(), lr=INIT_LR)
# opt.load_state_dict(previous_state['optimizer_state_dict'])

scheduler = ReduceLROnPlateau(opt, mode='min', factor=0.3, patience=2)
# scheduler.load_state_dict(previous_state['scheduler_state_dict'])

# initializez functia de eroarea si metrica R2
lossFn = nn.MSELoss()
r2score_metric = R2Score(multioutput='raw_values', device=device)

# initializez dictionarul istoricului costurilor
H = {
    "magnitude_train_loss":[],
    "latitude_train_loss":[],
    "longitude_train_loss":[],
    "depth_train_loss":[],
    "magnitude_val_loss":[],
    "latitude_val_loss":[],
    "longitude_val_loss":[],
    "depth_val_loss":[],
}

# incarc istoricul costurilor si ultima epoca - checkpoint
# H = previous_state['train_loss_history']
# last_epoch = previous_state['epoch']+1

# cronometrez durata procesului de antrenare
print("[INFO] training the network...")
startTime = time.time()

# initializez functia de Early Stopping
early_stopper = EarlyStopping(patience = 4)
```