Brain Anomaly Detection

Ghetoiu Gheorghe-Laurentiu

Aprilie 2023

1 Cuprins

2 Despre problema

Problema consta in separarea in doua clase a unor imagini de tip computer tomograf pe creier. Cele doua clase sunt reprezentate de imaginile care contin anomalii si cele care nu contin anomalii pe creier. Imaginile sunt in nuante de gri, avand dimensiunea de 224x224 pixeli. Setul de date este alcatuit din 22149 de imagini, dintre care 2000 sunt folosite pentru validare, iar 5149 sunt folosite pentru test.

3 Citirea si analizarea datelor

Pentru citire am folosit libraria PILLOW. Chiar daca imaginile sunt alb-negru, ele sunt citite ca si cum ar avea 3 canale, fiecare avand aceeasi valoare. Am transformat imaginile astfel incat sa ramana un singur canal si apoi le-am salvat intr-un numpy array caruia sa ii pot da load din motive de eficienta, citirea din fisier direct a imaginilor dureaza destul de mult.

In ceea ce priveste analizarea datelor, am observat ca exista imaginii complet negre sau aproape negre. Initial am eliminat aceste imagini, in functie de numarul de pixeli diferiti de 0 din imagine, dar apoi am observat ca astfel de imagini se afla si setul de validare si testare si am decis sa le pastrez considerand ca vor ajuta la prezicerea corecta si a acelor imagini din validare/test. De asemenea, analizand etichetele, mai ales dupa antrenarea catorva modele am observat ca metricile sunt mai bune pentru clasa 0, indicand un dezechilibru al claselor.

Clasa 0	Clasa 1
12762	**2238**

Table 1: Repartitia claselor pe train

4 KNN

Cei mai apropiati k vecini (k-nearest neighbors) este un algoritm de invatare, folosit atat in clasificare cat si in regresie. Pentru problema noastra, clasificare, algorimtul alege o eticheta pentru un input nou bazandu-se pe clasa majoritara dintre cei mai apropiati k vecini, unde k este un numar fixat. Pentru a calcula distanta dintre date se pot folosi diversi algoritmi cum ar fi distanta Euclidiana, distanta Manhattan sau distanta Minkowski. Apoi datele sunt sortate in ordinea asemanarii(distantei) dintre ele si sunt selectate cele k cu cea mai mica distanta. In caz ca mai multe imagini se afla la aceeasi distanta de imaginea noua, se alege la intamplare dintre acestea sau se folosesc alte criterii, unul dintre ele fiind alegerea dupa clasa care este deja majoritara.

precision	recall	f1-score	precision	recall	f1-score	nr_vecini/norm
Clasa 0			Clasa 1			
0.88	0.91	0.90	0.32	0.26	0.29	1/l1
0.88	0.90	0.89	0.29	0.26	0.27	1/l2
0.87	0.98	0.92	0.39	0.07	0.12	2/l1
0.87	0.96	0.88	0.40	0.09	0.15	2/l2
0.87	0.96	0.91	0.32	0.12	0.18	3/l1
0.81	0.92	0.89	0.30	0.1	0.15	3/l2
0.87	0.99	0.92	0.41	0.05	0.08	4/l1
0.86	0.99	0.92	0.33	0.01	0.03	15/l1

Table 2: Valori obtinute

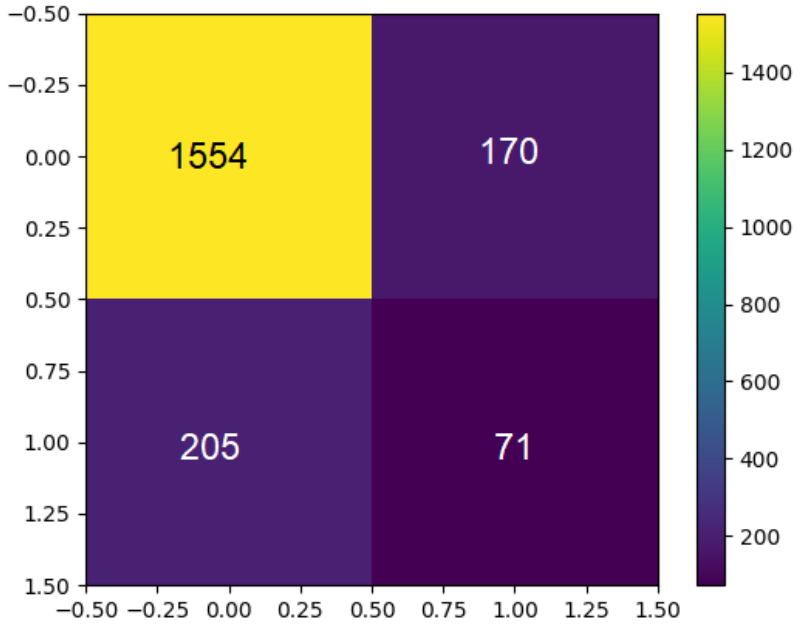

Figure 1: Matricea de confuzie pentru KNN

Pentru mai multi vecini KNN prezice foarte slab, mai ales pentru clasa 1, fiind clasa minoritara. Pentru acest model am pastrat varianta cu 1 vecin si normalizare l1, obtinand un f1 pe ambele clase de 0.31. De asemenea am incercat sa fac oversampling pentru clasa 1, dar nici acest lucru nu a imbunatatit acuratetea.

5 CNN

Retelele Neuronale Convolutionale sau CNN(Convolutional Neural Networks) sunt un tip de retele neuronale folosite mai ales pentru clasificarea imaginilor, dar si pentru recunoasterea video.

Fiind un tip de retele neuronale, acestea sunt bazate pe mai multe layere de neuroni care invata sa extraga trasaturi (relevante), in acest caz, din imagini. Primul layer este de cele mai multe ori un layer convolutional, care aplica o convolutie pe inputul dat si extrage trasaturi (la inceput linii, colturi). Rezultatul este apoi trecut printr-o functie de activare (de cele mai multe ori functii din

familia ReLU - Rectified Linear Unit function). Aceasta functie de activare este folosita pentru a introduce nonliniaritate in retea, spre exemplu ReLU seteaza valorile negative obtinute in urma convolutiei la 0.

Urmeaza apoi mai multe layere convolutionale, fiecare extragand trasaturi din ce in ce mai complexe, de regula cu cat se creste in adancime creste si numarul de filtre(campuri receptive) care invata trasatrui mai specifice/abstracte cum ar fi texturi, forme specifice, materiale.

In final sunt folosite si layere fully connected(complet conectate) care aplica regresie pe rezultatul convolutiilor, avand rolul sa ajunga la probabilitatile ca imaginea sa apartina uneia dintre clasele posibile.

Convolutia este o operatie care presupune combinarea a doua functii pentru a obtine una noua. In cadrul retelelor convolutionale, ne referim la input si la filtre (numite si kernels sau feature detectors). Operatia de convolutie presupune sa trecem *prin* filtru imaginea (defapt filtrul sa treaca peste imagine) facand o inmultire dintre matricea filtrului si bucata din imagine peste care se afla. Valoarea obtinua este gradul de similiritate dintre filtru si portiunea scanata din imagine.

Dupa unul sau mai multe layere convolutionale urmeaza, de regula, un layer de pooling (cel mai folosit fiind max pooling). In acest caz max pooling extrage cea mai mare valoare din rezultat(matricea obtinuta) pastrand astfel cele mai importante atribute.

5.1 Primul model incercat

Primul model incercat

Pentru a citi pozele am folosit PILLOW. Pozele sunt transformate in grayscale si sunt salvate intr-un numpy array pe care l-am folosit mai apoi pentru a incarca datele direct din el. Ca metode de normlaizare pentru primul model, dar si pentru toate celelalte, am impartit fiecare pixel din imagine la 255 pentru a ramane cu valori intre 0 si 1. Functia de activare folosita este **reLU** pentru toate modelele, iar pentru ultimul layer am folosit **sigmoid** avand ca output probabilitatea pentru clasa 1.

type	filters	kernel
convolutional	16	(3,3)
MaxPooling(2,2)		
convolutional	32	(3,3)
MaxPooling(2,2)		
convolutional	64	(3,3)
convolutional	128	(3,3)
convolutional	256	(3,3)
MaxPooling(2,2)		
Flatten()		
dense	256	
dense	1	

Table 3: Modelul incercat

Pentru acest model am experimentat doar cu learning rate-ul, batch size si numarul de epoci.

Learning Rate	Batch Size	Nr. Epochs	recall	precision	f1	accuracy
0.0001	32	10	0.24	0.68	0.35	0.88
0.0001	64	10	0.37	0.64	0.47	0.885
0.0001	64	20	0.34	0.64	0.44	0.881
0.0001	32	20	0.39	0.61	0.47	0.88
0.0001	128	20	0.355	0.662	0.462	0.879
0.00001	32	30	0.25	0.621	0.356	0.862
0.0001	32	30	0.32	0.669	0.528	0.89

Table 4: Valori obtinute

precision	recall	f1-score	precision	recall	f1-score
Clasa 0			Clasa 1		
0.89	0.98	0.93	0.68	0.24	0.36
0.91	0.97	0.94	0.65	0.38	0.48
0.90	0.97	0.93	0.65	0.34	0.45
0.91	0.96	0.93	0.62	0.39	0.48
0.90	0.97	0.94	0.66	0.36	0.46
0.89	0.98	0.93	0.62	0.25	0.36
0.90	0.97	0.94	0.67	0.33	0.44

Table 5: Valori obtinute

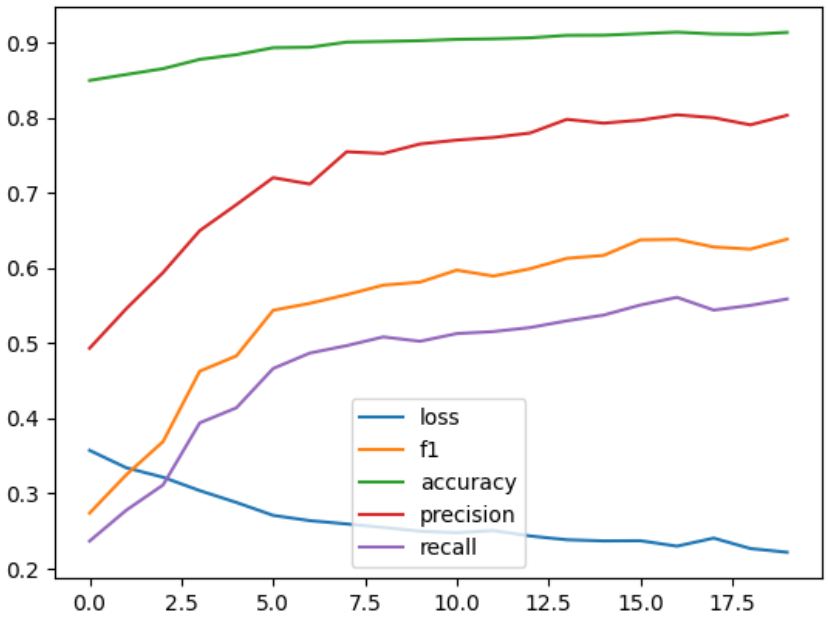

Figure 2: Evolutia modelului

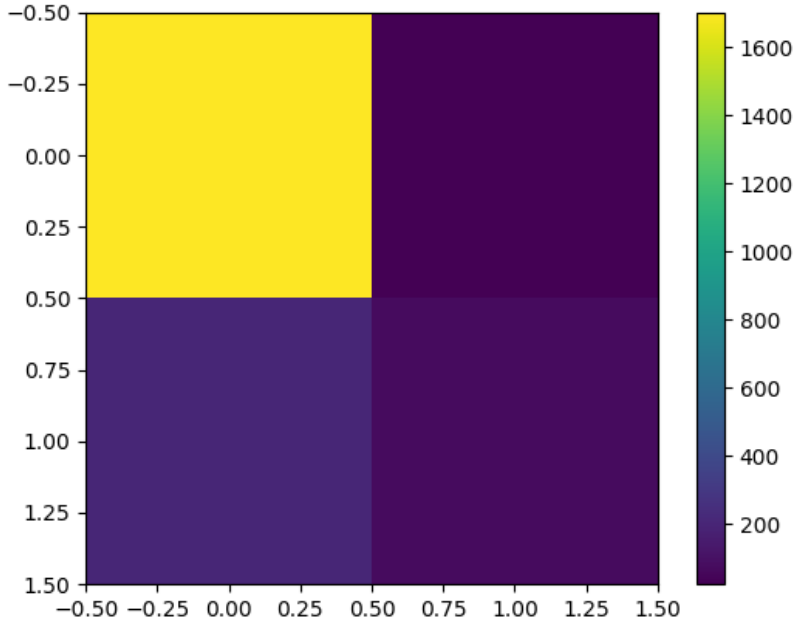

Figure 3: Matricea de confuzie

Cel mai bun rezultat este obtinut pe batch size de 64, 20 de epoci si learning rate 0.0001. Pentru mai multe epoci modelul face overfitting, iar f1 score-ul incepe sa scada, in mare parte din cauza ca scade recall-ul. Precizia este mai mare pentru batch-size mai mare, iar recall-ul este mai mic. Pentru learning rate mic modelul nu invata suficient de bine, nici pe mai multe epoci.

Concluzii
Batch size-ul intre 32 si 64 obtine cele mai bune rezultate, modelul este destul de simplu si nu face overfit in prima faza, deci poate sa fie antrenat undeva pana la 30-40 de epoci. Se observa in tabelul care contine metricile pe clase ca pentru clasa 0, modelul prezice semnificativ mai bine decat pentru clasa 1, asta duncandu-ne cu gandul la faptul ca clasele nu sunt balansate.

5.2 Al doilea model incercat

Aici am folosit aceeasi metoda pentru a incarca imaginile ca mai sus. Diferentele pentru acest model constau in faptul ca am crescut complexitatea adaugand mai multe layere, cu mai multe filtre. Pe langa asta am folosit mai multe tehnici de augumentare (cum ar fi central crop, rotatii ale imaginilor, zoom). Pentru a preveni overfitting-ul am adaugat dropout is batch normalization. Am experimentat si cu diversi kernel initializers, dar si cu valori diferite pentru dropout si tehnicile de augumentare. De asemenea, paddingul este setat ca ”same” pentru toate layerele, iar functia de activare folosita este ”reLU”, mai putin pentru stratul de output, unde am folosit ”sigmoid”.

Table 6: Modelul incercat

type	filters	kernel
convolutional	32	(3,3)
Batch Normalization		
convolutional	64	(3,3)
Batch Normalization		
convolutional	64	(3,3)
Batch Normalization		
MaxPooling - 2x2		
convolutional	128	(3,3)
Batch Normalization		
convolutional	128	(3,3)
Batch Normalization		
convolutional	128	(3,3)
Batch Normalization		
MaxPooling - 2x2		
convolutional	256	(3,3)
Batch Normalization		
convolutional	256	(3,3)
Batch Normalization		
convolutional	256	(3,3)
Batch Normalization		
convolutional	256	(3,3)
Batch Normalization		
MaxPooling - 2x2)		
Dropout - 0.35		
Flatten()		
dense	512	
dense	512	
Droput - 0.15		
dense	1	

Dropout-ul este folosit ca tehnica de regularizare pentru a preveni supra-antrenarea retelei(overfitting). Acesta functioneaza setand la 0 valorile anumitor neuroni alesi la intamplare. Dropout-ul ajuta prin mai multi factori. In primul rand, deoarece la fiecare iteratie sunt folosti alti neuroni, acestia devin mai putin dependenti unii de alti, permitandu-i modelului sa invete trasaturi mai generale. Tot din acest motiv, droput-ul are si rol de ensabmble-learning, practic obtinand in fiecare epoca un model putin diferit, avand neuroni diferiti.

Batch normalization, dupa cum ii spune si numele normalizeaza datale, aducandu-le la medie 0 si varianta 1, dar o aplica rezultatului obtinut in urma functiei de activare la nivel de batch. Aceasta reduce si nevoia de dropout prin faptul ca adauga zgomot in date, ajutand la prevenirea overifitting-ului. De asemenea imbunatateste si timpul de invatare (modelul invata mai repede). Acesta reprezinta si motivul pentru care am renuntat la dropout intre layerele convoltuionale, deoarece am observat ca nu aduc imbunatatiri.

Tehnicile de augumentare sunt folosite tot pentru a preveni overfitting-ul, acestea ajutand atat la obtinerea unor caracteristici mai relevante, cat si introducand zgomot in imagini. Experimentad cu augumentarea, aceasta aduce imbunatatiri destul de mici in ceea priveste f1 score, dar permit modelului sa fie antrenat mai mult, dar pentru a nu strica acuratetea a trebuit sa folosesc

valori care aduc modificari mici. Printre tehnicile de augumentare folosite se numara rotatia random, care roteste imaginile cu an anumit grad, flip dar pe care am renuntat sa il folosesc deoarece nu aducea imbunatatiri, zoom random, pentru care am setat un factor foarte mic.

In prima faza am folosit learning rate constant, si nu am folosit nicio tehnica de augumentare, obtinand urmatoarele rezultate.

precision	recall	f1-score	precision	recall	f1-score	acc	nr. epoci
Clasa 0			Clasa 1				
0.87	0.93	0.90	0.84	0.32	0.46	0.88	10
0.91	0.95	0.93	0.88	0.38	0.54	0.89	20
0.90	0.98	0.94	0.79	0.20	0.32	0.89	40

Table 7: Valori obtinute

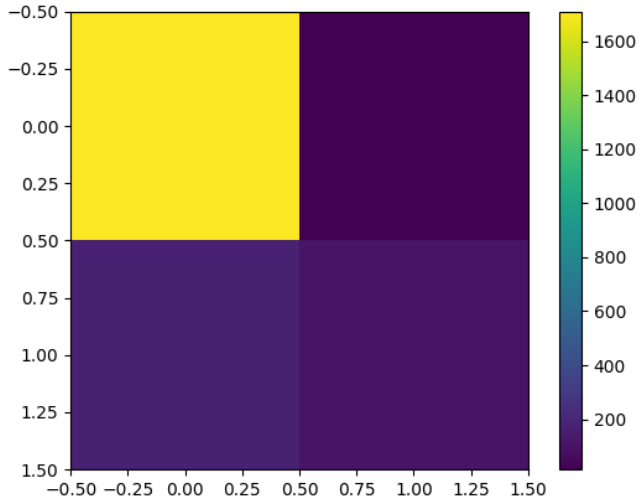

Figure 4: Matricea de confuzie

Dupa cate se poate observa, acest model deja incepe sa faca overfitting pe mai mult de 20 de epoci, prezicand insa foarte bine pe imaginile din clasa majoritara, dar mai prost pe imaginile din clasa minoritara. Cel mai bun rezultat pe acest model a fost obtinut pe 20 de epoci si a avut un f1_score de 0.53 pe validare.

Pentru urmatoarea antrenare am adaugat si class weight, obtinand urmatorul rezulat.

precision	recall	f1-score	precision	recall	f1-score	acc	nr. epoci
Clasa 0			Clasa 1				
0.92	0.91	0.91	0.40	0.70	0.51	0.87	10
0.96	0.83	0.89	0.43	0.79	0.56	0.87	20
0.95	0.92	0.93	0.52	0.68	0.59	0.88	30
0.86	0.99	0.93	0.85	0.20	0.32	0.89	50

Table 8: Valori obtinute

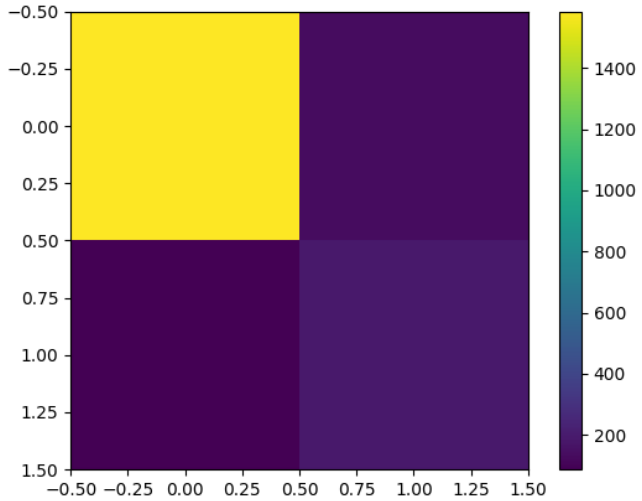

Figure 5: Matricea de confuzie

Cu toate ca modelul obtine rezultate mai bune, in special in ceea ce priveste clasa minoritara, acesta tot este predispus spre overfitting. Din aceasta cauza am adaugat augumentare. Pentru urmatoarea antrenare, deoarece am folosit mai multe epoci, am adaugat si learning rate care se schimba dinamic, scazand pe parcurs ce modelul este antrenat de mai multe epoci.

Rezultatele obtinute astfel sunt cele mai bune pe acest model, avand un f1_score general pe validare de 0.685.

precision	recall	f1-score	precision	recall	f1-score	acc	nr. epoci
Clasa 0			Clasa 1				
0.96	0.85	0.90	0.46	0.68	0.55	0.89	30
0.91	0.92	0.91	0.53	0.62	0.57	0.88	50
0.95	0.92	0.93	0.58	0.63	0.60	0.90	70
0.94	0.95	0.94	0.62	0.68	0.65	0.91	100

Table 9: Valori obtinute

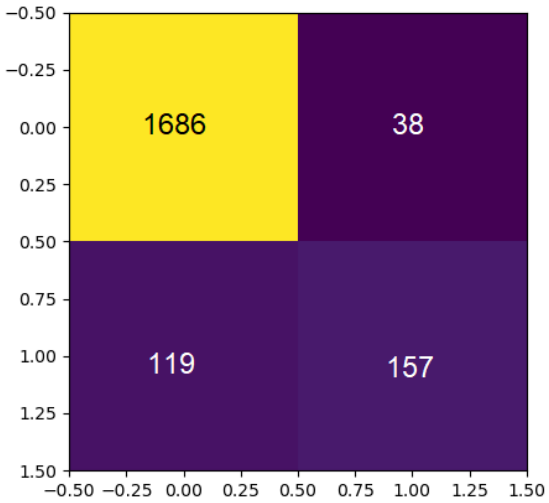

Figure 6: Matricea de confuzie

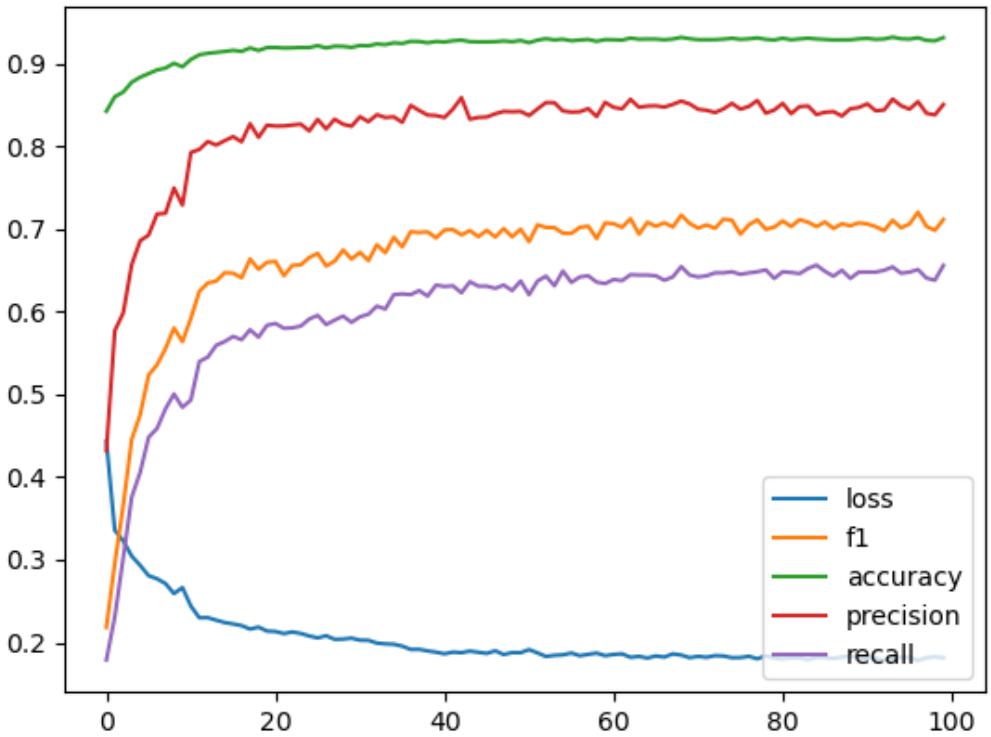

Figure 7: Evolutia modelului

5.3 Al treila model incercat

Pentru al treilea model de CNN, am folosit si layer skipping. Aceasta tehnica consta in crearea unor scurtaturi intre straturi de la niveluri mai inalte la straturi mai adanci, astfel sarind anumite straturi. Acest lucru rezolva problema prin care gradientii devin prea mici, limitand adancimea pe care poate sa o aiba modelul. De asemenea imbunatateste si viteza de antrenare a modelului,

acesta convergand mai repede la o solutie. Pentru a crea un bloc cu skip connection, putem sa avem doua sau mai multe layere convolutionale, dar pentru unul dintre ele nu aplicam convolutia pe rezultatul stratului anterior, ci pe rezultatul unui layer aflat mai sus in ierarhie. La final adunam rezultatele dintre cele doua layere care formeaza skip connection-ul.

type	filters	input	output
convolutional	x	$input_0$	rez_1
convolutional	x	rez_1	rez_2
convolutional	x	$input_0$	rez_3
suma	$rez2 + rez_3$		rez_f

Table 10: Structura unui bloc

In prima faza am incercat sa antrenez un model, dar complexitatea acestuia era prea mare si nu ajungea sa invete. Am inceput cu 5 layere cu 64 de filtre si am crescut numarul layerelor cu 1 pe masura ce am dublat filtrele.

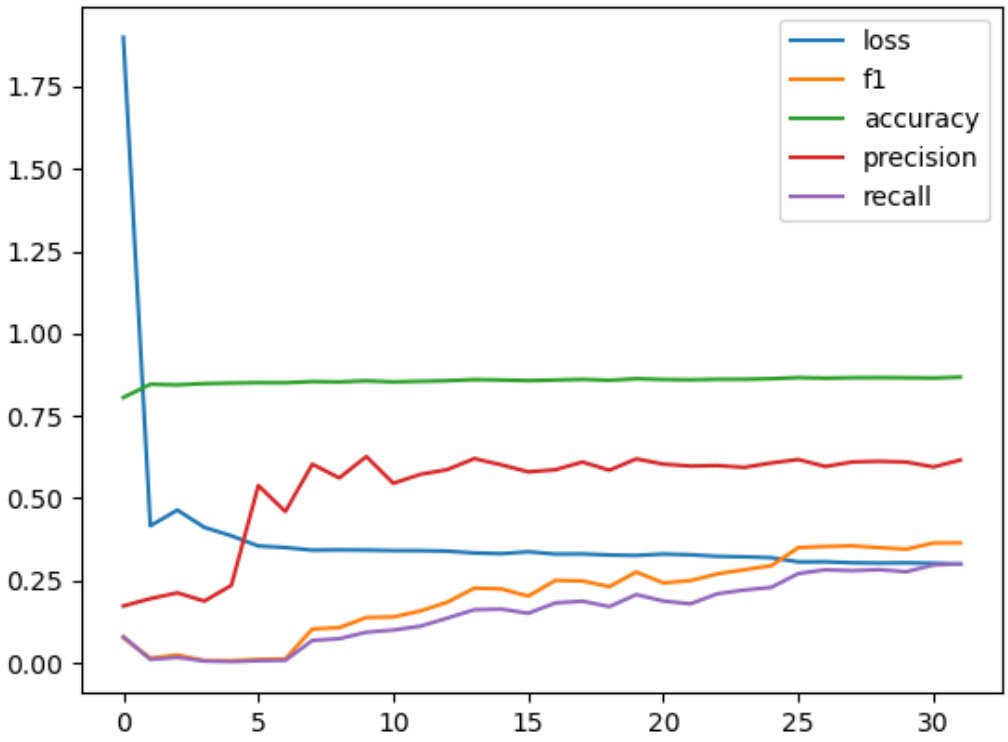

Figure 8: Modelul nu invata

Un model mai bun este alcatuit din 3 blocuri cu 64 de filtre, 4 blocuri cu 128 si 5 cu 256 de filtre. In final am aplicat average pooling si inca doua straturi fully-conected cu un output de 512, urmate de stratul de output care este acelasi ca pentru celelalte modele, 1 output si functie de activare 'sigmoid'.

Pentru acest model am observat ca face overfitting multe mai repede, motiv pentru care am adaugat pentru fiecare layere si un kernel regularizer, cel mai bun rezultat l-am obtinut pentru l2 (regresie ridge) cu valoarea 0.001. Pentru augumentare am folosit aceleasi tehinci ca mai la modelul anterior. De asemenea, dupa cum se poate observa in figura de mai jos, f1 score variaza

destul de mult, avand niste spik-uri in anumite momente, dar modelul converge pana la urma, stabilizandu-se. Pentru acest model am ales sa folosesc pentru predictii una dintre epocile care obtine un f1 bun pe validare, dar care nu este nici foarte departata de f1 pe train. Maximul obtinut cu acest model pe validare este f1: 0.667.

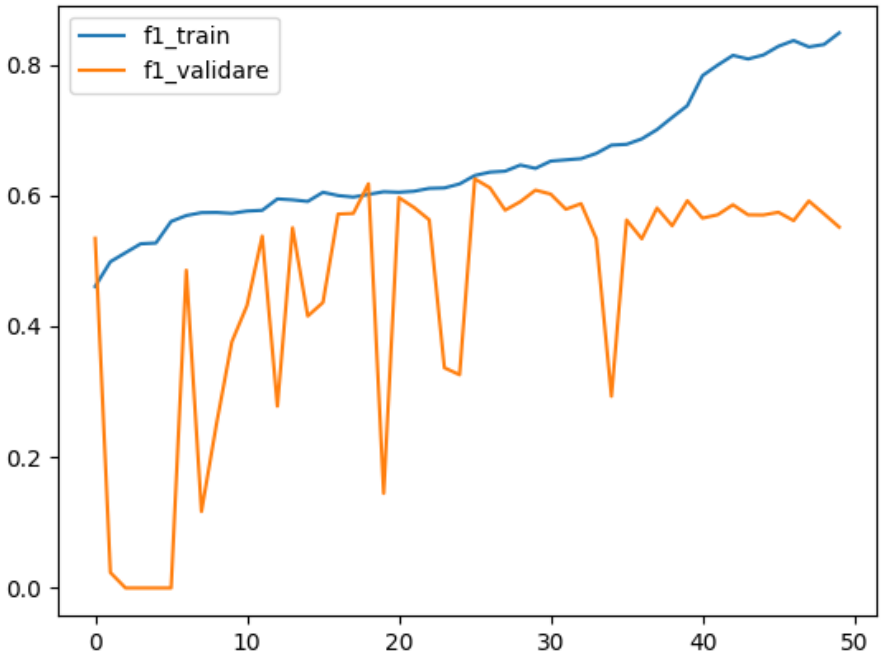

Figure 9: Diferenta dintre f1 pe train si f1 pe validare

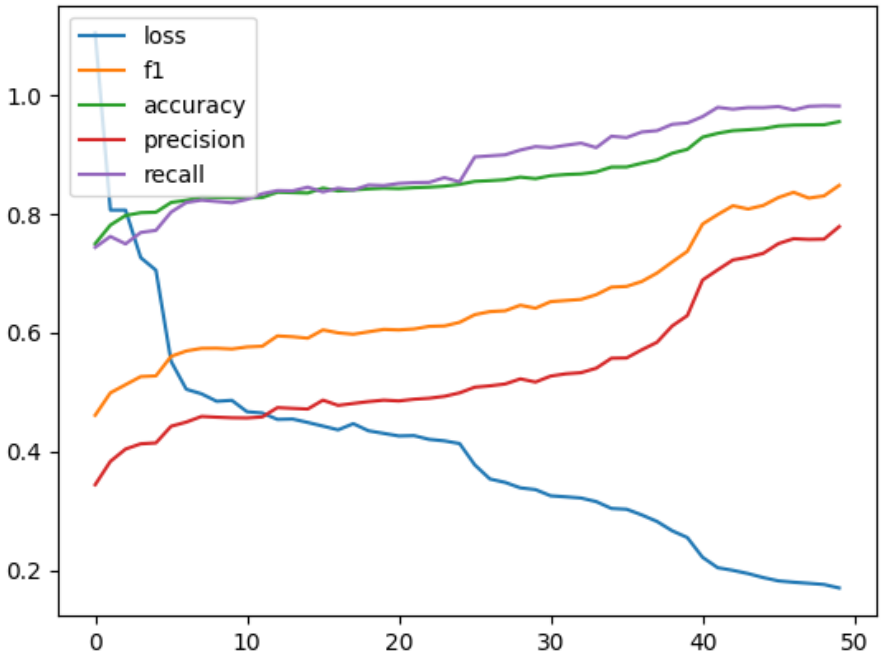

Figure 10: Evolutia modelului

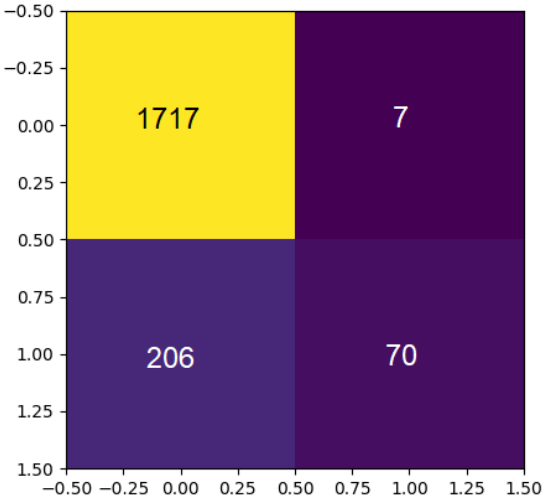

Figure 11: Matricea de confuzie

6 Bibliografie

Documentatie tensorflow: `https://www.tensorflow.org/api_docs/python/tf/all_symbols`
Documentatie keras: `https://keras.io/api/`
Documentatie pillow: `https://pillow.readthedocs.io/en/stable/reference/Image.html`
Cursuri si laboratoare: `https://fmi-unibuc-ia.github.io/ia/`