

UNIVERSITATEA BUCUREŞTI
Facultatea de Matematică și Informatică
Programul de studii: *Informatică*



**UNIVERSITATEA DIN
BUCUREŞTI**
— VIRTUTE ET SAPIENTIA —

Disciplina: Inteligență Artificială

***Documentație
Proiect Laborator
- Competiție „Deep Hallucination Classification”-***

Student:

Ştefania RÎNCU

Grupa 232

**BUCUREŞTI
2023**

Cuprins

Model 1 – K-NN	3
Descrierea celui mai bun model	3
Modele încercate	6
Rezultate în funcție de parametrii	6
Model 2 – CNN	7
Preprocesarea datelor	7
Primul model testat	7
Adăugarea checkpoint-ului	9
Adăugarea LearningRateScheduler-ului	9
Adăugarea EarlyStopping-ului.....	9
Modelul final	11
Hyperparameter tuning	17
LearningRate-uri testate	17
Optimizatori testați.....	20
Numărul de epoci.....	21
Hiperparametrii testați	21
Alte arhitecturi de CNN încercate	21
Observație importantă.....	25
Mentire.....	25
Concluzii	29

Model 1 - K-NN

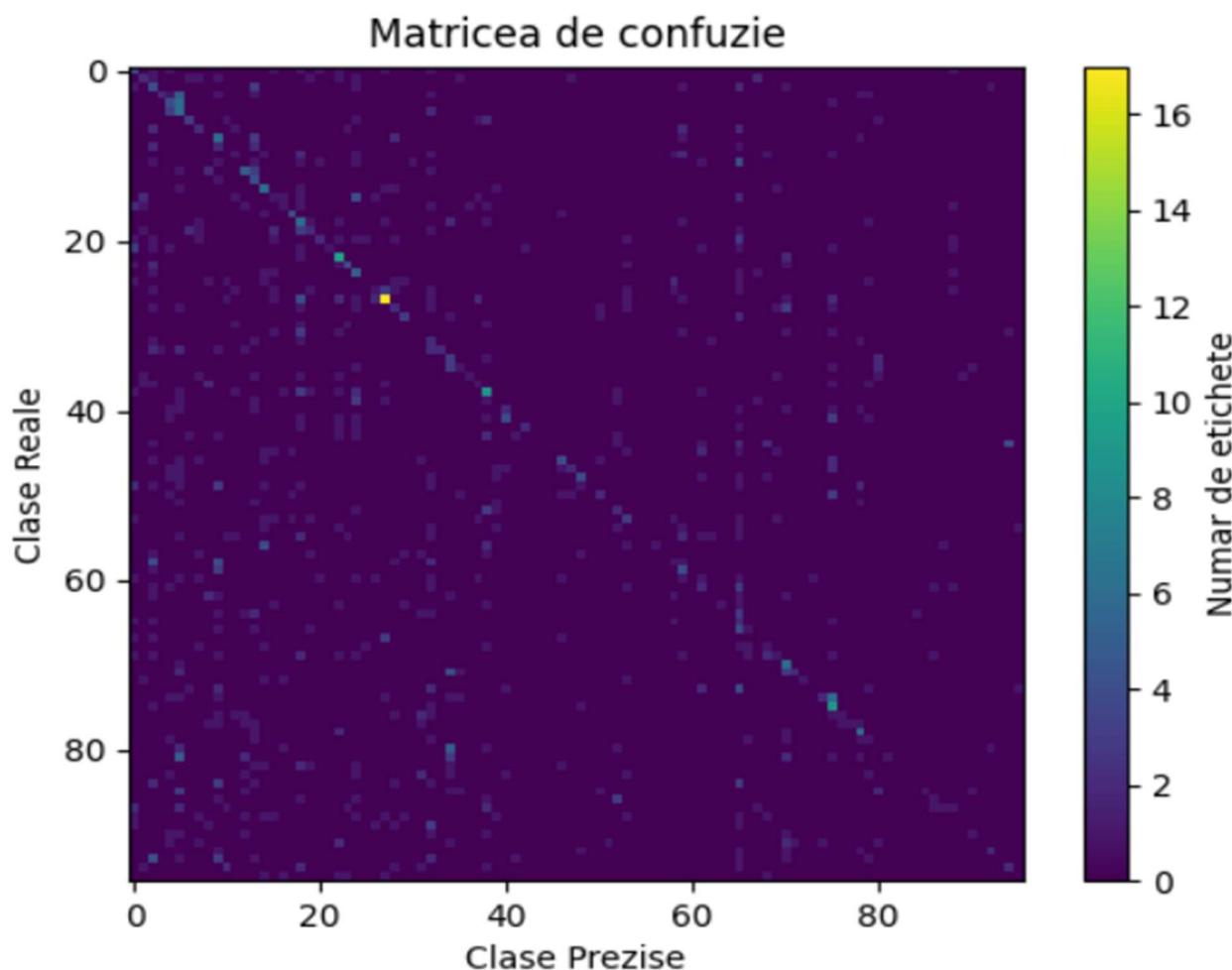
Primul model pe care l-am în testat în cadrul aceste competiții a fost un model de tip K-NN, care clasifică imaginile pe baza celor mai apropiati K vecini.

Descrierea celui mai bun model

Pentru început, pentru preprocesarea datelor, am preluat pe rând fiecare imagine și am transformat-o într-un vector de pixeli, pentru care am calculat media și deviația standard, ca mai apoi să pot standardiza imaginea respectivă. De asemenea, în momentul în care am adăugat imaginea standardizată în vectorul de imagini, am transformt-o într-un tablou unidimensional.

În varianta, care a obținut cel mai bun scor, am setat numărul de vecini selectați la fiecare pas să fie egal cu 10 și metrica folosită pentru calcularea distanței să fie de tipul „cityblock”. Această distanță de tip „cityblock” este echivalentă cu distanța „manhattan”. Valoarea ei este egală cu suma modulelor diferențelor dintre doi vectori, mai exact în cazul nostru, pentru două imagini

Pentru această clasificare am obținut o acuratețe maximă de 19.5%.

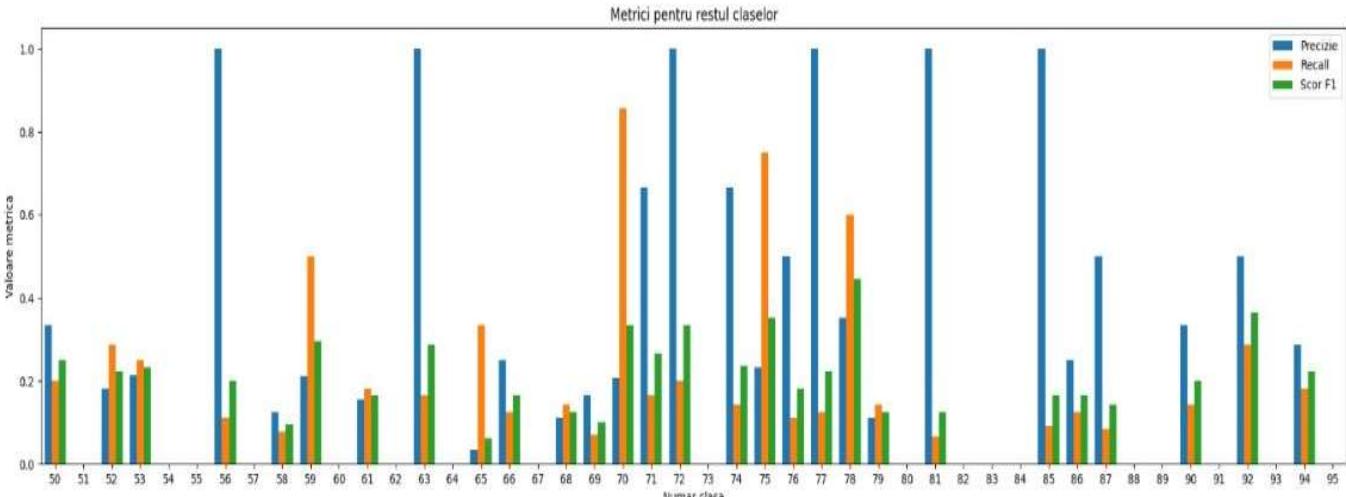
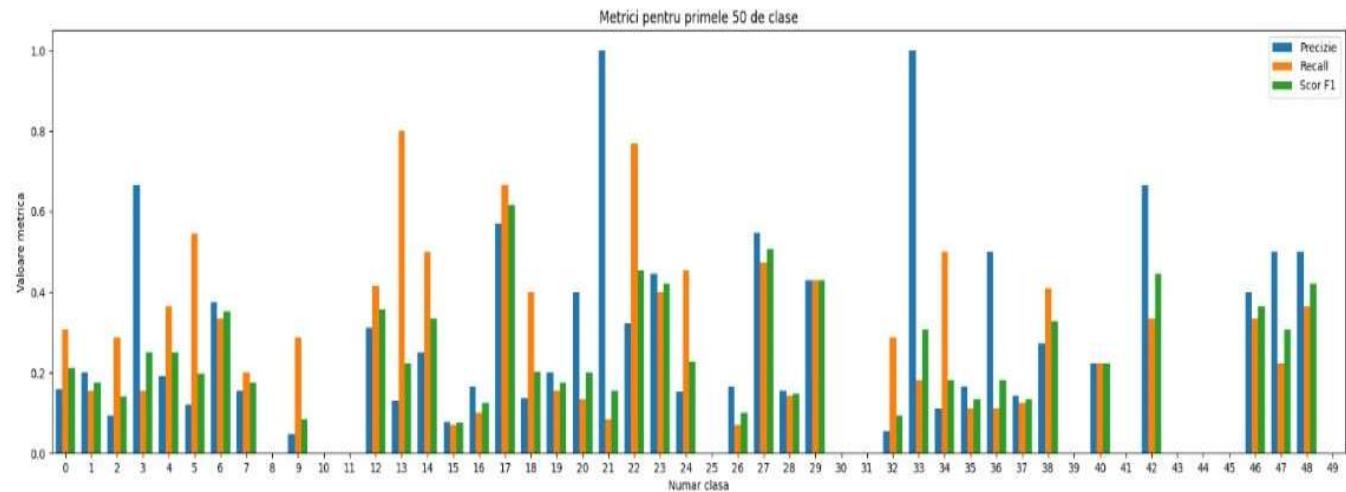


Statistici pentru fiecare clasa:

Clasa	Adevarat Pozitive	Fals Pozitive	Fals Negative	Precizie	Recall	F1
0	4	21	9	0.160	0.308	0.211
1	2	8	11	0.200	0.154	0.174
2	4	39	10	0.093	0.286	0.140
3	2	1	11	0.667	0.154	0.250
4	4	17	7	0.190	0.364	0.250
5	6	44	5	0.120	0.545	0.197
6	3	5	6	0.375	0.333	0.353
7	2	11	8	0.154	0.200	0.174
8	0	9	13	0.000	0.000	0.000
9	2	39	5	0.049	0.286	0.083
10	0	7	11	0.000	0.000	0.000
11	0	8	12	0.000	0.000	0.000
12	5	11	7	0.312	0.417	0.357
13	4	27	1	0.129	0.800	0.222
14	6	18	6	0.250	0.500	0.333
15	1	12	13	0.077	0.071	0.074
16	1	5	9	0.167	0.100	0.125
17	4	3	2	0.571	0.667	0.615
18	6	38	9	0.136	0.400	0.203
19	2	8	11	0.200	0.154	0.174
20	2	3	13	0.400	0.133	0.200
21	1	0	11	1.000	0.083	0.154
22	10	21	3	0.323	0.769	0.455
23	4	5	6	0.444	0.400	0.421
24	5	28	6	0.152	0.455	0.227
25	0	2	10	0.000	0.000	0.000
26	1	5	13	0.167	0.071	0.100
27	17	14	19	0.548	0.472	0.507
28	2	11	12	0.154	0.143	0.148
29	3	4	4	0.429	0.429	0.429
30	0	1	4	0.000	0.000	0.000
31	0	9	9	0.000	0.000	0.000
32	2	34	5	0.056	0.286	0.093
33	2	0	9	1.000	0.182	0.308
34	3	24	3	0.111	0.500	0.182
35	1	5	8	0.167	0.111	0.133
36	1	1	8	0.500	0.111	0.182
37	1	6	7	0.143	0.125	0.133
38	9	24	13	0.273	0.409	0.327
39	0	7	9	0.000	0.000	0.000
40	2	7	7	0.222	0.222	0.222
41	0	2	13	0.000	0.000	0.000
42	2	1	4	0.667	0.333	0.444
43	0	0	8	0.000	0.000	0.000
44	0	0	11	0.000	0.000	0.000
45	0	0	8	0.000	0.000	0.000
46	4	6	8	0.400	0.333	0.364
47	2	2	7	0.500	0.222	0.398
48	4	4	7	0.500	0.364	0.421
49	0	0	14	0.000	0.000	0.000
50	2	4	8	0.333	0.200	0.250
51	0	1	4	0.000	0.000	0.000
52	2	9	5	0.182	0.286	0.222
53	3	11	9	0.214	0.250	0.231
54	0	0	8	0.000	0.000	0.000
55	0	0	12	0.000	0.000	0.000
56	1	0	8	1.000	0.111	0.200
57	0	0	7	0.000	0.000	0.000
58	1	7	12	0.125	0.077	0.095
59	4	15	4	0.211	0.500	0.296
60	0	0	13	0.000	0.000	0.000
61	2	11	9	0.154	0.182	0.167
62	0	2	8	0.000	0.000	0.000
63	1	0	5	1.000	0.167	0.286
64	0	0	10	0.000	0.000	0.000
65	2	57	4	0.034	0.333	0.062
66	1	3	7	0.250	0.125	0.167

67	0	5	6	0.000	0.000	0.000
68	1	8	6	0.111	0.143	0.125
69	1	5	13	0.167	0.071	0.100
70	6	23	1	0.207	0.857	0.333
71	2	1	10	0.667	0.167	0.267
72	1	8	4	1.000	0.200	0.333
73	0	1	14	0.000	0.000	0.000
74	2	1	12	0.667	0.143	0.235
75	9	30	3	0.231	0.750	0.353
76	1	1	8	0.500	0.111	0.182
77	1	0	7	1.000	0.125	0.222
78	6	11	4	0.353	0.600	0.444
79	1	8	6	0.111	0.143	0.125
80	0	9	11	0.000	0.000	0.000
81	1	0	14	1.000	0.067	0.125
82	0	0	8	0.000	0.000	0.000
83	0	0	9	0.000	0.000	0.000
84	0	1	11	0.000	0.000	0.000
85	1	0	10	1.000	0.091	0.167
86	1	3	7	0.250	0.125	0.167
87	1	1	11	0.500	0.083	0.143
88	0	8	12	0.000	0.000	0.000
89	0	1	8	0.000	0.000	0.000
90	1	2	6	0.333	0.143	0.200
91	0	0	11	0.000	0.000	0.000
92	2	2	5	0.500	0.286	0.364
93	0	0	15	0.000	0.000	0.000
94	2	5	9	0.286	0.182	0.222
95	0	2	6	0.000	0.000	0.000

Metrici per-clasă pentru predicțiile făcute asupra datelor de validare



Reprezentarea procentelor pentru precizie, recall și scor f1 pentru fiecare clasă

Modele încercate

Inițial, nu am standardizat imaginile, ci doar le-am normalizat, împărțind valorile pixelilor acestora la 255.0, astfel încât aceștia să aibă o valoare între 0.0 și 1.0. Cu această modalitate de augmentare, pentru aceleiași parametrii ($k = 10$ și distanța ‘cityblock’), acuratețea modelului ajungea la 15.5%.

De asemenea, am testat modelul, atribuind diferite metriki pentru distanțe și diferite valori pentru contorul celor mai apropiati K vecini.

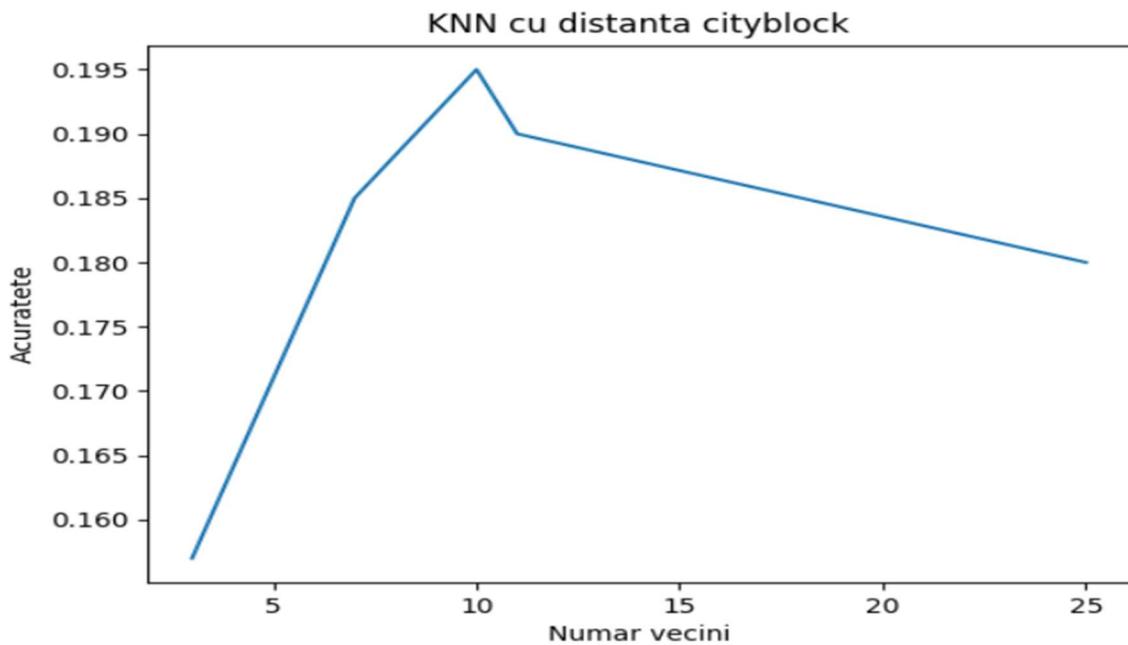
Rezultate în funcție de parametrii

METRICI		REZULTAT
Numărul vecinilor selectați	Metrica pentru distanță	Acuratețe model
3	l_2	12.8%
10	l_2	13.7%
25	l_2	14.3%
3	cosine	12.8%
5	cosine	13.3%
25	cosine	14.3%
100	cosine	12.6%
3	cityblock	15.7%
7	cityblock	18.5%
11	cityblock	19%
10	cityblock	19.5%
25	cityblock	18%

Tipurile de distanțe pe care le-am testat:

- l_2 – distanța euclidiană
- cosine - este o distanță care determină similitudinea dintre două imagini, și este calculată astfel: avem doi vectori, pentru care înmulțim elementele aflate pe aceeași poziție și însumăm aceste produse; pentru fiecare vector calculăm suma pătratelor fiecărui element și apoi determinăm radicalul acestei valori; suma obținută la primul pas o împărțim la produsul dintre valorile radicalilor obținuți la cel de-al doilea pas
- cityblock (sau manhattan) – distanța pe care am și selectat-o ca fiind cea mai bună

Am observat faptul că, atunci când selectez puțini vecini acuratețea nu este suficient de bună, și am incrementat treptat numărul vecinilor. Totuși, dacă adaug prea mulți vecini, acuratețea începe să scadă dintr-un anumit punct.



Graful asociat acurateței modelului în funcție de numărul de vecini, folosind distanța „cityblock”

Model 2 – CNN

Pentru cel de-al doilea model testat în cadrul competiției, am implementat o arhitectură de rețea neuronală conoluțională. În acest proces, am realizat mai multe modele care au eșuat și am schimbat mai multe arhitecturi, am testat diverse hiperparametri și am folosit diferite augmentări și preprocesări ale datelor, până când am obținut o versiune a unui model cu o rată de acuratețe mai mare.

Preprocesarea datelor

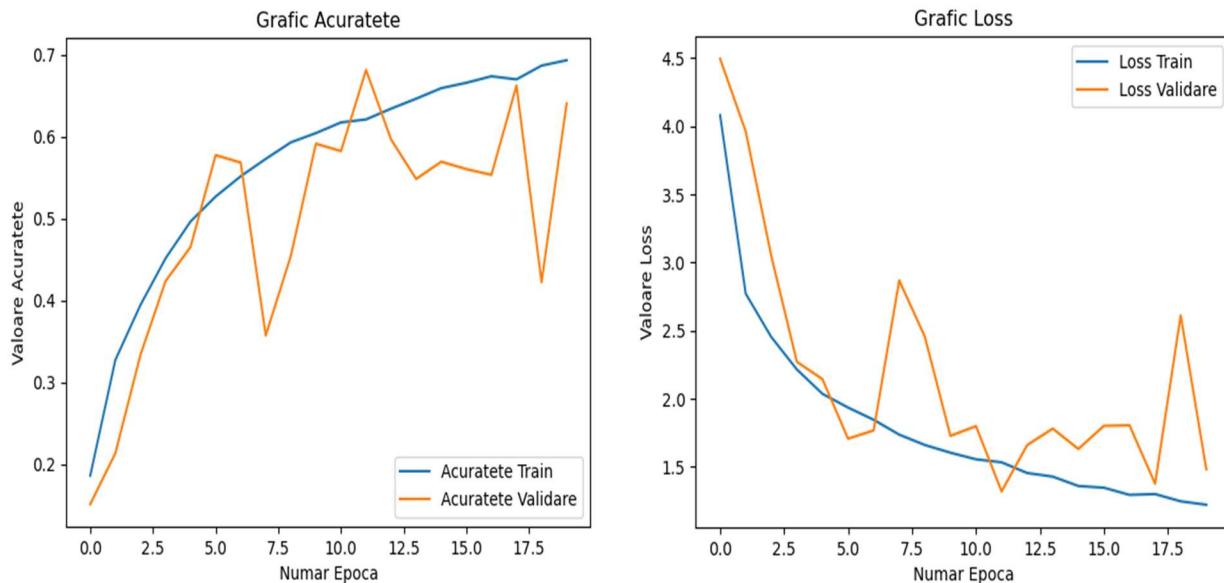
Pentru început, pentru preprocesarea datelor, am preluat pe rând fiecare imagine, și am redimensionat-o, astfel încât să mă asigur că toate imaginile vor avea aceeași dimensiune ca cea a imputului dat ca parametru în cadrul primului strat conoluțional. Apoi, am transformat fiecare imagine într-un vector de pixeli, pe care l-am normalizat împărțind valorile din cadrul acestuia la 255.0, astfel încât fiecare pixel să se afle în cadrul intervalului determinat de 0.0 și 1.0. De asemenea, în cadrul vectorilor care rețin etichetele am adăugat valorile întregi ale acestora. Această preprocesare a labelurilor este necesară pentru a putea folosi funcția de activare „softmax” în cadrul ultimui strat „fully connected” din cadrul modelului.

Primul model testat

Primul model pe care l-am testat a fost format din patru straturi conoluționale și două straturi dense, dintre care ultimul folosit pentru clasificare. Toate straturile conoluționale erau construite după tiparul: (Conoluție, BatchNormalization și MaxPooling, de 2x2), și nu am folosit deloc Dropout. Pentru augmentarea imaginilor, am folosit librăria „ImageDataGenerator”, și am adăugat asupra acestora rotiri, shiftări și zoom-uri care arău aplicate în intervale mici. Learning rate-ul pe care l-am aplicat la început a

fost unul constant, și anume am declarat un optimizator „Adam”, căruia i-am setat diverse learning rate-uri în intervalul 0.001 și 0.01. Numărul de epoci pentru antrenare nu a depășit valoarea 30, iar dimensiunea loturilor pentru antrenare am setat-o la 16, valoare care am descoperit ulterior că este mult prea mică. Aceste modele simpliste nu a trecut de o acuratețe de 65%.

Layer (type)	Output Shape	Param #
conv2d_54 (Conv2D)	(None, 62, 62, 64)	1792
batch_normalization_71 (Batch Normalization)	(None, 62, 62, 64)	256
max_pooling2d_40 (MaxPooling2D)	(None, 31, 31, 64)	0
conv2d_55 (Conv2D)	(None, 29, 29, 64)	36928
batch_normalization_72 (Batch Normalization)	(None, 29, 29, 64)	256
max_pooling2d_41 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_56 (Conv2D)	(None, 12, 12, 128)	73856
batch_normalization_73 (Batch Normalization)	(None, 12, 12, 128)	512
max_pooling2d_42 (MaxPooling2D)	(None, 6, 6, 128)	0
conv2d_57 (Conv2D)	(None, 2, 2, 256)	819456
batch_normalization_74 (Batch Normalization)	(None, 2, 2, 256)	1024
max_pooling2d_43 (MaxPooling2D)	(None, 1, 1, 256)	0
flatten_10 (Flatten)	(None, 256)	0
dense_27 (Dense)	(None, 128)	32896
batch_normalization_75 (Batch Normalization)	(None, 128)	512
dense_28 (Dense)	(None, 96)	12384
<hr/>		
Total params:	979,872	
Trainable params:	978,592	
Non-trainable params:	1,280	



Putem observa din grafice că acest prim model implementat avea creșteri și scăderi brusăe în ceea ce privește atât acuratețea, cât și valoarea loss-ului pentru datele de validare. De asemenea, se vede ușor faptul că, atunci când se încheie etapa de antrenare, modelul păstrează structura pe care a avut-o în cadrul ultimei epoci, când a obținut o acuratețe mai mică decât valoarea maximă atinsă în cadrul întregului proces de antrenare. Atunci am căutat o soluție de a avea certitudinea că la final voi avea structura modelului care a făcut cele mai bune clasificări (similar cu algoritmii genetici unde aleg cel mai bun individ).

Adăugarea checkpoint-ului

Înțial, am crezut că modelul nu reușește să își îmbunătățească performanța, deoarece nu îl las la antrenat suficiente epoci, dar oricăr de mult îl lăsăm nu reușeam să obțin valori mai bune, ba chiar le înrăutăteam. De câteva ori, am avut diferențe mari între valoarea maximă a acurateții atinse pe validare în timpul antrenării și valoarea obținută la final, deoarece modelul obținea o acuratețe foarte mică chiar în ultima epocă.

Astfel, pentru a obține rezultate mai bune, următorul pas a fost să adaug modelului un „callback”, în cadrul căruia să salvez în timpul antrenării cea mai bună versiune a acestuia. Funcția din keras care se ocupă cu acest checkpoint primește ca parametrii o cale, care determină fișierul în care salvăm modelul. Eu am ales să salvez numai cea mai bună versiune a modelului, care este aleasă în funcție de acuratețea pe datele de validare. Nu aleg să salvez doar ponderile acestuia, ci păstrează întregul model, deoarece astfel se va păstra doar cel mai bun și nu va fi suprascris de valori anterioare. Cum în cadrul competiției scorul era obținut în funcție de acuratețe, am încercat să maximizez această valoare, și de aceea, am setat ca și criteriu pentru monitorizare scorul acurateței pe datele de validare. După încheierea etapei de antrenare, preiau modelul salvat și pe baza acestuia fac predicțiile finale, având certitudinea că este cel mai bun din punct de vedere al acurateței.

Adăugarea LearningRateScheduler-ului

Prin adăugarea checkpoint-ului am reușit să rezolv o problemă: și anume aveam la final cel mai bun model și reușeam să fac predicții mai bune. Totuși, dintr-un anumit punct, modelul nu mai învăța noi caracteristici și chiar dacă îl lăsăm la antrenat pe o durată foarte lungă, nu obțineam rezultate mai bune. Atunci am ajuns la concluzia că trebuie să modific cumva rata de învățare pe parcursul antrenării. Astfel, am adăugat un „scheduler”, care să „programeze” rata de învățare a modelului și să o adapteze în funcție de epocă. Acest scheduler l-am definit printr-o lambda expresie, pe care am definit-o ca fiind egală cu rata inițială de învățare, multiplicată cu o rată de scădere dată ca hiperparametru, pe care o ridicăm la puterea rezultatului împărțirii dintre epoca actuală și numărul total de epoci. Astfel, am simulaat o rată de învățare cu un „exponential decay”.

```
callback_rata_invatare = callbacks.LearningRateScheduler(  
    lambda ep: param_rata_initiala_lr * param_rata_scadere ** (ep + param_epoci))
```

Adăugarea EarlyStopping-ului

Văzând că am reușit să îmbunătățesc acuratețea modelului cu ajutorul ratei de învățare adaptabile, m-am gândit să obțin un model și mai bun, antrenând modelul de mai multe ori pe câte o epocă, și de fiecare dată când intru într-o nouă etapă de antrenare să preiau modelul salvat și să îl reantrenez. Astfel, am obținut un

scor destul de bun. Dar testând, am văzut că dacă îl las mai multe epoci în cadrul funcției de fit, acuratețea crește mai mult. Atunci am venit cu o nouă soluție.

Am adăugat o instrucțiune repetitivă în cadrul căreia să apelez funcția de antrenare de mai multe ori, pentru un număr mai mare de epoci. Își de fiecare dată să reantrenez cel mai bun model.

```
for ep in range(hiper_epoci_mari):
    print('\n----- Epoch: {} / {} -----'.format(ep + 1, hiper_epoci_mari))
    cnn_clasificator.antreneaza_cel_mai_bun(imagini_train, labeluri_train, imagini_validare, labeluri_validare,
                                                param_rata_initiala_lr=hiper_rata_initiala_lr, param_rata_scadere=hiper_rata_scadere,
                                                param_dimens_lot=hiper_dimens_lot, param_epoci=hiper_epoci_train)

----- Epoch: 1 / 25 -----
Epoch 1/20
749/750 [=====>.] - ETA: 0s - loss: 6.1878 - accuracy: 0.1600
Epoch 1: val_accuracy improved from -inf to 0.24300, saving model to best_cnn.h5
750/750 [=====] - 35s 46ms/step - loss: 6.1854 - accuracy: 0.1600 - val_loss: 4.1811 - val_accuracy: 0.2430 - lr: 4.4200e-04
Epoch 2/20
749/750 [=====>.] - ETA: 0s - loss: 3.1097 - accuracy: 0.3570
Epoch 2: val_accuracy improved from 0.24300 to 0.28500, saving model to best_cnn.h5
750/750 [=====] - 34s 45ms/step - loss: 3.1089 - accuracy: 0.3572 - val_loss: 3.0448 - val_accuracy: 0.2850 - lr: 4.2432e-04
Epoch 3/20
750/750 [=====] - ETA: 0s - loss: 2.2810 - accuracy: 0.4498
Epoch 3: val_accuracy improved from 0.28500 to 0.48700, saving model to best_cnn.h5
750/750 [=====] - 34s 46ms/step - loss: 2.2810 - accuracy: 0.4498 - val_loss: 2.0026 - val_accuracy: 0.4870 - lr: 4.0735e-04
Epoch 4/20
750/750 [=====] - ETA: 0s - loss: 1.9478 - accuracy: 0.5017
Epoch 4: val_accuracy improved from 0.48700 to 0.54400, saving model to best_cnn.h5
750/750 [=====] - 34s 45ms/step - loss: 1.9478 - accuracy: 0.5017 - val_loss: 1.7918 - val_accuracy: 0.5440 - lr: 3.9106e-04
Epoch 5/20
749/750 [=====>.] - ETA: 0s - loss: 1.7407 - accuracy: 0.5431
Epoch 5: val_accuracy did not improve from 0.54400
750/750 [=====] - 35s 46ms/step - loss: 1.7405 - accuracy: 0.5429 - val_loss: 2.8206 - val_accuracy: 0.3450 - lr: 3.7541e-04
Epoch 6/20
749/750 [=====>.] - ETA: 0s - loss: 1.5973 - accuracy: 0.5774
Epoch 6: val_accuracy did not improve from 0.54400
750/750 [=====] - 35s 47ms/step - loss: 1.5978 - accuracy: 0.5772 - val_loss: 2.1748 - val_accuracy: 0.4520 - lr: 3.6040e-04
Epoch 7/20
749/750 [=====>.] - ETA: 0s - loss: 1.5094 - accuracy: 0.5971
Epoch 7: val_accuracy did not improve from 0.54400
750/750 [=====] - 34s 46ms/step - loss: 1.5089 - accuracy: 0.5973 - val_loss: 2.3745 - val_accuracy: 0.4170 - lr: 3.4598e-04
Epoch 8/20
749/750 [=====>.] - ETA: 0s - loss: 1.4093 - accuracy: 0.6250
Epoch 8: val_accuracy did not improve from 0.54400
750/750 [=====] - 34s 46ms/step - loss: 1.4091 - accuracy: 0.6250 - val_loss: 2.5638 - val_accuracy: 0.3560 - lr: 3.3214e-04
Epoch 9/20
750/750 [=====] - ETA: 0s - loss: 1.3276 - accuracy: 0.6475
Epoch 9: val_accuracy improved from 0.54400 to 0.56100, saving model to best_cnn.h5
750/750 [=====] - 35s 47ms/step - loss: 1.3276 - accuracy: 0.6475 - val_loss: 1.6689 - val_accuracy: 0.5610 - lr: 3.1886e-04
Epoch 10/20
749/750 [=====>.] - ETA: 0s - loss: 1.2692 - accuracy: 0.6586
Epoch 10: val_accuracy did not improve from 0.56100
750/750 [=====] - 34s 45ms/step - loss: 1.2691 - accuracy: 0.6586 - val_loss: 2.6083 - val_accuracy: 0.3880 - lr: 3.0610e-04
Epoch 11/20
749/750 [=====>.] - ETA: 0s - loss: 1.2157 - accuracy: 0.6676
Epoch 11: val_accuracy improved from 0.56100 to 0.67900, saving model to best_cnn.h5
750/750 [=====] - 34s 46ms/step - loss: 1.2163 - accuracy: 0.6675 - val_loss: 1.1759 - val_accuracy: 0.6790 - lr: 2.9386e-04
Epoch 12/20
750/750 [=====] - ETA: 0s - loss: 1.1639 - accuracy: 0.6863
Epoch 12: val_accuracy improved from 0.67900 to 0.68000, saving model to best_cnn.h5
750/750 [=====] - 35s 46ms/step - loss: 1.1639 - accuracy: 0.6863 - val_loss: 1.2172 - val_accuracy: 0.6800 - lr: 2.8210e-04
Epoch 13/20
750/750 [=====] - ETA: 0s - loss: 1.1323 - accuracy: 0.6913
```

```

----- Epoch: 9 / 25 -----
Epoch 1/28
749/750 [=====>>>] - ETA: 0s - loss: 0.7531 - accuracy: 0.7968
Epoch 1: val_accuracy did not improve from 0.84980
750/750 [=====] - 34s 45ms/step - loss: 0.7538 - accuracy: 0.7966 - val_loss: 1.0381 - val_accuracy: 0.7340 - lr: 4.4200e-04
Epoch 2/28
750/750 [=====] - ETA: 0s - loss: 0.7573 - accuracy: 0.8023
Epoch 2: val_accuracy did not improve from 0.84980
750/750 [=====] - 48s 53ms/step - loss: 0.7573 - accuracy: 0.8023 - val_loss: 1.2480 - val_accuracy: 0.6900 - lr: 4.2432e-04
Epoch 3/28
750/750 [=====] - ETA: 0s - loss: 0.7189 - accuracy: 0.8154
Epoch 3: val_accuracy did not improve from 0.84980
750/750 [=====] - 48s 54ms/step - loss: 0.7189 - accuracy: 0.8154 - val_loss: 0.7788 - val_accuracy: 0.8050 - lr: 4.0735e-04
Epoch 4/28
750/750 [=====] - ETA: 0s - loss: 0.7176 - accuracy: 0.8123
Epoch 4: val_accuracy did not improve from 0.84980
750/750 [=====] - 48s 53ms/step - loss: 0.7176 - accuracy: 0.8123 - val_loss: 1.2834 - val_accuracy: 0.6930 - lr: 3.9106e-04
Epoch 5/28
750/750 [=====] - ETA: 0s - loss: 0.7039 - accuracy: 0.8189
Epoch 5: val_accuracy did not improve from 0.84980
750/750 [=====] - 48s 53ms/step - loss: 0.7039 - accuracy: 0.8189 - val_loss: 1.1693 - val_accuracy: 0.7100 - lr: 3.7541e-04
Epoch 6/28
750/750 [=====] - ETA: 0s - loss: 0.6982 - accuracy: 0.8191
Epoch 6: val_accuracy did not improve from 0.84980
750/750 [=====] - 41s 54ms/step - loss: 0.6982 - accuracy: 0.8191 - val_loss: 0.9175 - val_accuracy: 0.7720 - lr: 3.6040e-04
Epoch 7/28
749/750 [=====>>>] - ETA: 0s - loss: 0.6789 - accuracy: 0.8223
Epoch 7: val_accuracy did not improve from 0.84980
750/750 [=====] - 39s 52ms/step - loss: 0.6786 - accuracy: 0.8224 - val_loss: 0.7382 - val_accuracy: 0.8120 - lr: 3.4598e-04
Epoch 8/28
749/750 [=====>>>] - ETA: 0s - loss: 0.6633 - accuracy: 0.8300
Epoch 8: val_accuracy did not improve from 0.84980
750/750 [=====] - 48s 53ms/step - loss: 0.6642 - accuracy: 0.8298 - val_loss: 0.8564 - val_accuracy: 0.7840 - lr: 3.3214e-04
Epoch 9/28
750/750 [=====] - ETA: 0s - loss: 0.6388 - accuracy: 0.8310
Epoch 9: val_accuracy did not improve from 0.84980
750/750 [=====] - 48s 54ms/step - loss: 0.6388 - accuracy: 0.8310 - val_loss: 1.0547 - val_accuracy: 0.7430 - lr: 3.1886e-04
Epoch 10/28
749/750 [=====>>>] - ETA: 0s - loss: 0.6332 - accuracy: 0.8336
Epoch 10: val_accuracy did not improve from 0.84980
750/750 [=====] - 48s 53ms/step - loss: 0.6344 - accuracy: 0.8333 - val_loss: 0.9275 - val_accuracy: 0.7600 - lr: 3.0610e-04
Epoch 11/28
750/750 [=====] - ETA: 0s - loss: 0.6217 - accuracy: 0.8346
Epoch 11: val_accuracy did not improve from 0.84980
750/750 [=====] - 41s 55ms/step - loss: 0.6217 - accuracy: 0.8346 - val_loss: 1.0195 - val_accuracy: 0.7320 - lr: 2.9386e-04
Epoch 12/28
749/750 [=====>>>] - ETA: 0s - loss: 0.6172 - accuracy: 0.8390
Epoch 12: val_accuracy did not improve from 0.84980
750/750 [=====] - 41s 55ms/step - loss: 0.6173 - accuracy: 0.8389 - val_loss: 0.7254 - val_accuracy: 0.8120 - lr: 2.8210e-04

```

După cum se poate observa, în epoca 9, modelul deja nu mai reușea să își mai îmbunătățeasă performanțele. Cauza principală fiind aceea că la intrarea în fiecare etapă nouă de antrenare se resetă rata de învățare. Și totuși, am așteptat până la finalul ultimei epoci ca să pot avea predicțiile, fiind cel mai bun scor obținut până în acel moment. Atunci am adăugat un alt „callback” pentru o oprire timpurie, în funcție de cum variază acuratețea sau loss-ul modelului pe validare.

La început, am setat ca și criteriu pentru monitorizare, „val_accuracy”, iar modelul era oprit dacă nu evoluă din punct de vedere al acurateței pe validare în cadrul procesului de antrenare. Ulterior, când am schimbat diverse rate de învățare am setat monitorizarea pentru „val_loss”, deoarece aveam momente în care modelul nu avea performanțe cea timp, dar apoi creștea brusc acuratețea. Apoi am revenit la „val_accuracy”.

Modelul final

Arhitectura rețelei finale, conține un total de șase straturi convoluționale și trei dense, dintre care ultimul este folosit pentru clasificarea finală a imaginilor. Acest model este implementat folosind Sequential, care facilitează conectarea straturilor rețelei neurale, dar care impune și o constrângere, și anume faptul că fiecare strat admite un singur input și un singur output. Această structurare a modelului a fost foarte utilă din perspectiva faptului că permite serializare pentru salvarea celui mai bun model.

- Asupra tuturor straturilor (mai puțin a ultimului dens) aplic funcția de activare „relu” (Rectified Linear Unit), care are proprietatea că va întoarce maximul dintre 0 și suma weighturilor din input
- Pentru ultimul strat, folosesc ca activare funcția „softmax”, care realizază o distribuție a probabilităților ca o imagine dată să se afle în fiecare clasă din cele 96
- În cadrul rețelei sunt prezente două straturi de dropout cu o rată de 0.3, astfel încât să previn supraînvățarea modelului. Introducând aceste straturi, anumiți neuroni sunt eliberați, atribuindu-le valoarea 0, astfel se elimină eroarea de generalizare, care duce la overfitting

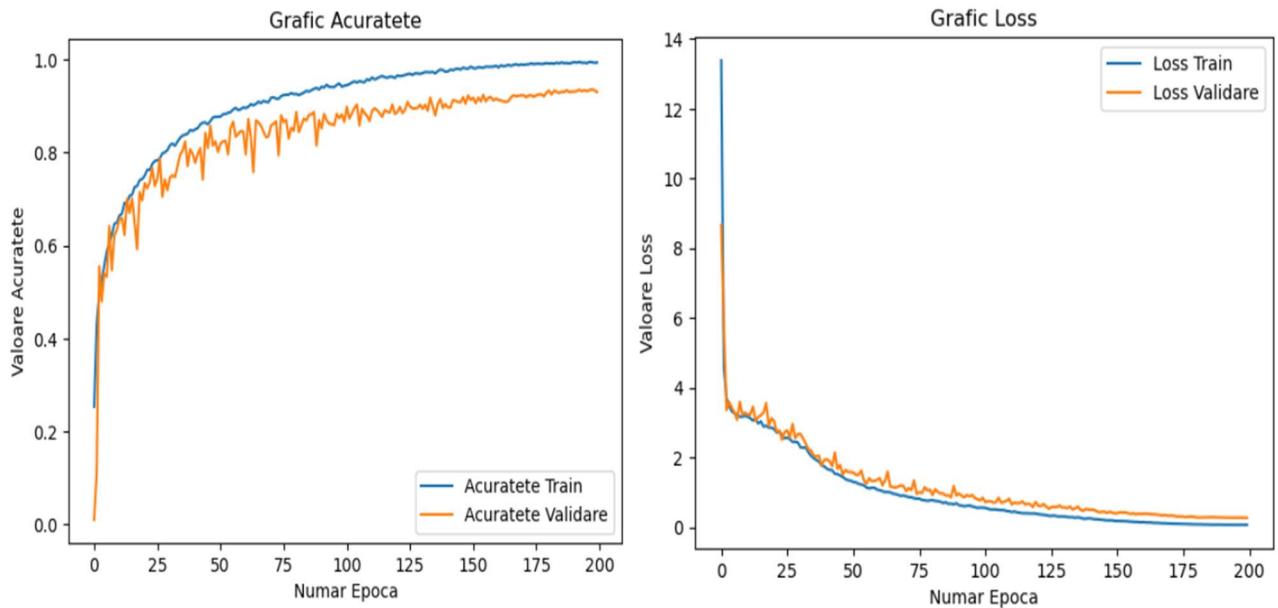
- Asupra primelor două straturi dense aplic o regularizare a kernelului, de tip l2. Astfel, aplic penalizări în timpul optimizării. L2, presupune faptul că se înmulțește valoarea setată de mine (0.035) cu suma radicalelor inputurilor.
- Dimensiunea kernelului aplicat asupra fiecărui strat conoluțional este de (3x3), deoarece am încercat să adaug kernel-uri de dimensiuni mai mari, dar de cele mai multe ori am eșuat prin a nu se potrivi dimensiunile finale
- În cadrul rețelei sunt prezente trei straturi de max pooling care să restrângă dimensiunile straturilo și care aplică o matrice de 2x2 peste stratul conoluțional și selectează valoarea maximă dintre cele patru pătrate pe care le acoperă
- Numărul maxim de filtre folosite este de 512, iar cel minim pe 32, în cadrul primului strat. Am adăugat puține filtre în primele straturi pentru a facilita învățarea, iar în ultimele straturi am adăgat cât mai multe filtre
- În etapa de compilare a modelului, pentru calcularea loss-ului folosesc funcția „SparseCategoricalCrossentropy()”, și de aceea în etapa de preprocesare transform etichetele în valori întregi. Ca și optimizator, folosesc „Adam”, căruia îi aplic un „weight-decay” destul de mic (0.001), care reușește să adapteze pondările și gradienții mai bine decât clasicul SGD.
- Ca și callback-uri am inclus: un checkpoint din care să pot prelua cel mai bun model obținut după încheierea etapei de antrenare pe parcursul tuturor epocilor, și unde se salvează cel mai bun model din punct de vedere al loss-ului pe validare; un early-stopping, care să opreasă modelul după 30 de epoci în care acuratețea pe validare nu s-a îmbunătățit și un learning rate scheduler care să adapteze rata de invățare în funcție de epoca actuală
- Pentru scheduler-ul ratei de învățare am simulaț un „Cosine Annealing Decay”, care să favorizeze scăderea ratei într-un mod mai puțin brusc
- În cadrul procesului de augmentare a imaginilor, acestea sunt întoarse atât vertical, cât și orizontal, iar suplimentar, am simulaț o funcție în cadrul căreia să se genereze o valoare random, uniformă, între 0 și 1, iar dacă această valoare este mai mare decât o probabilitate dată să se stergă un dreptunghi de dimensiuni obținute random din cadrul imaginii. Mai exact, în funcție de aria imaginii și două valori random, care reprezintă o rată a aspectului (L:l) și o constantă cu care să multiplicăm aria, determinăm o porțiune, în cadrul imaginii, pe care să o înlocuim cu alți pixeli, generații aleator

```

    ...
    HIPERMARAMETRII
    hiper_epoci_train = 200
    hiper_rata_dropout = 0.3
    hiper_rata_initiala_invatare = 0.001
    hiper_rata_scadere = 0.3
    hiper_regularizare_kernel = 0.035
    hiper_ingeduinta = 30
    hiper_dimens_lot = 48

```

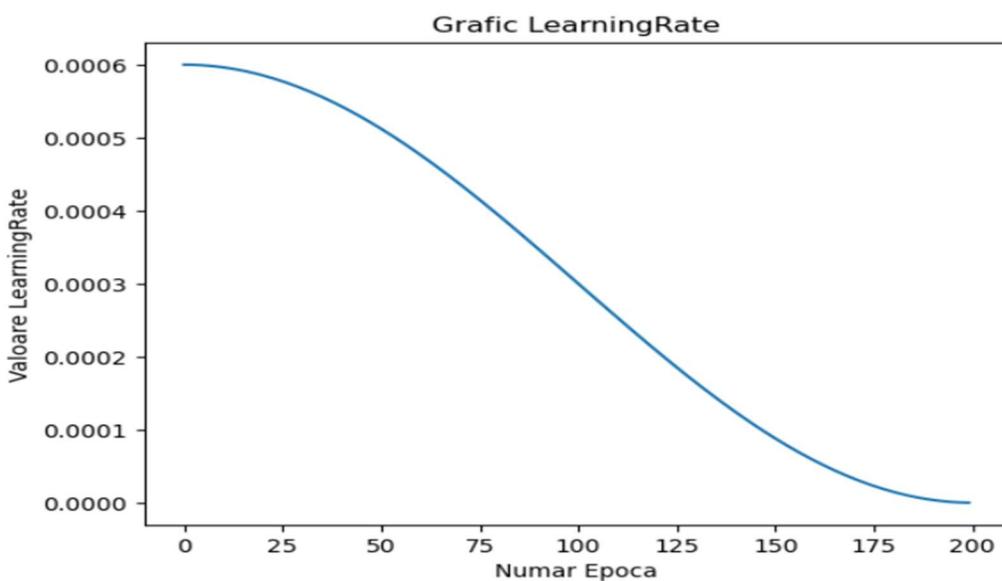
Valorile hiperparametrilor finali



Grafcile asociate acurateței și loss-ului

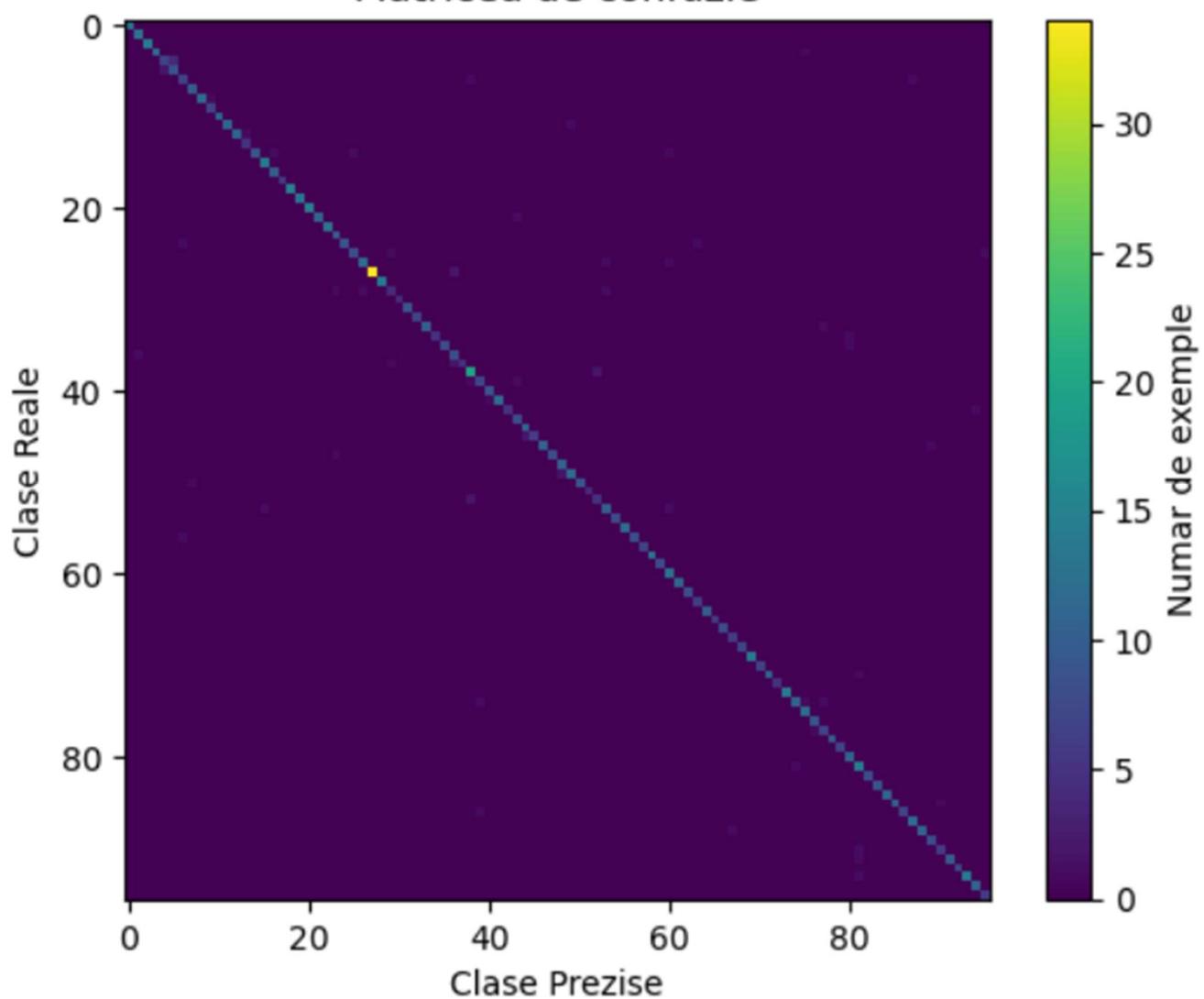
```
''' callback pentru care definesc un scheduler al ratei de învățare care să simuleze un Cosine Annealing LR '''
callback_cosine_annealing_lr = callbacks.LearningRateScheduler(
    lambda ep: param_rata_initiala_lr * param_rata_scdere * (1 + math.cos((math.pi * ep) / param_epoci)))
```

Funcția care definește comportamentul ratei de învățare (bazată pe funcția cosinus)



Graficul asociat comportamentului ratei de învățare (bazată pe funcția cosinus)

Matricea de confuzie



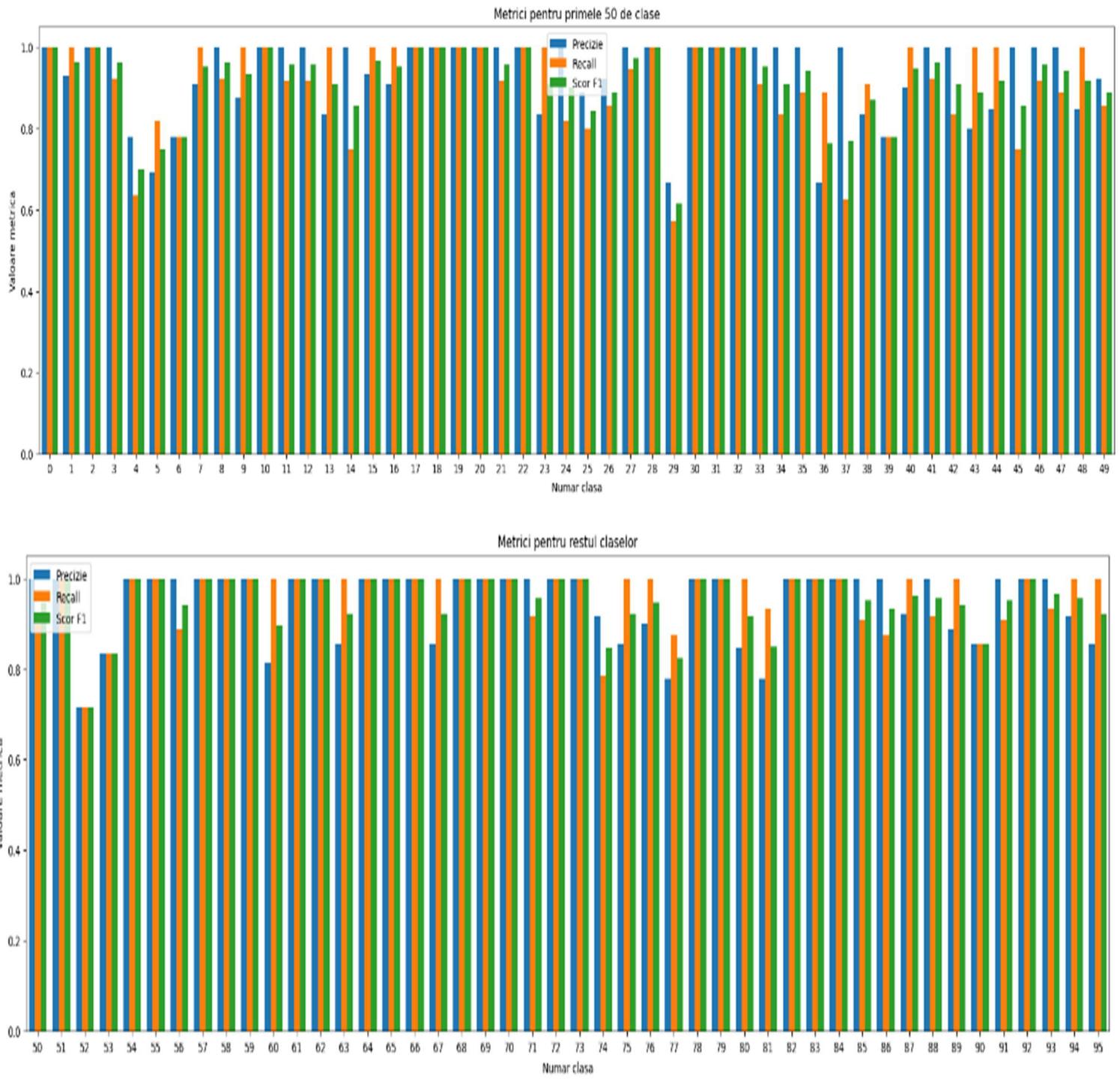
Matricea de confuzie asociată

Acuratete totală: 93.6000000000001%

Statistică pentru fiecare clasa:

Clasa	Adevarat Pozitive	Fals Pozitive	Fals Negative	Precizie	Recall	F1
0	13	0	0	1.000	1.000	1.000
1	13	1	0	0.929	1.000	0.963
2	14	0	0	1.000	1.000	1.000
3	12	0	1	1.000	0.923	0.960
4	7	2	4	0.778	0.636	0.700
5	9	4	2	0.692	0.818	0.750
6	7	2	2	0.778	0.778	0.778
7	10	1	0	0.909	1.000	0.952
8	12	0	1	1.000	0.923	0.960
9	7	1	0	0.875	1.000	0.933
10	11	0	0	1.000	1.000	1.000
11	11	0	1	1.000	0.917	0.957
12	11	0	1	1.000	0.917	0.957
13	5	1	0	0.833	1.000	0.909
14	9	0	3	1.000	0.750	0.857
15	14	1	0	0.933	1.000	0.966
16	10	1	0	0.909	1.000	0.952
17	6	0	0	1.000	1.000	1.000
18	15	0	0	1.000	1.000	1.000
19	13	0	0	1.000	1.000	1.000
20	15	0	0	1.000	1.000	1.000
21	11	0	1	1.000	0.917	0.957
22	13	0	0	1.000	1.000	1.000
23	10	2	0	0.833	1.000	0.909
24	9	0	2	1.000	0.818	0.900
25	8	1	2	0.889	0.800	0.842
26	12	1	2	0.923	0.857	0.889
27	34	0	2	1.000	0.944	0.971
28	14	0	0	1.000	1.000	1.000
29	4	2	3	0.667	0.571	0.615
30	4	0	0	1.000	1.000	1.000
31	9	0	0	1.000	1.000	1.000
32	7	0	0	1.000	1.000	1.000
33	10	0	1	1.000	0.909	0.952
34	5	0	1	1.000	0.833	0.909
35	8	0	1	1.000	0.889	0.941
36	8	4	1	0.667	0.889	0.762
37	5	0	3	1.000	0.625	0.769
38	20	4	2	0.833	0.909	0.870
39	7	2	2	0.778	0.778	0.778
40	9	1	0	0.900	1.000	0.947
41	12	0	1	1.000	0.923	0.960
42	5	0	1	1.000	0.833	0.909
43	8	2	0	0.800	1.000	0.889
44	11	2	0	0.846	1.000	0.917
45	6	0	2	1.000	0.750	0.857
46	11	0	1	1.000	0.917	0.957
47	8	0	1	1.000	0.889	0.941
48	11	2	0	0.846	1.000	0.917
49	12	1	2	0.923	0.857	0.889
50	9	0	1	1.000	0.900	0.947
51	4	0	0	1.000	1.000	1.000
52	5	2	2	0.714	0.714	0.714
53	10	2	2	0.833	0.833	0.833
54	8	0	0	1.000	1.000	1.000
55	12	0	0	1.000	1.000	1.000
56	8	0	1	1.000	0.889	0.941
57	7	0	0	1.000	1.000	1.000
58	13	0	0	1.000	1.000	1.000
59	8	0	0	1.000	1.000	1.000
60	13	3	0	0.812	1.000	0.897
61	11	0	0	1.000	1.000	1.000
62	8	0	0	1.000	1.000	1.000
63	6	1	0	0.857	1.000	0.923
64	10	0	0	1.000	1.000	1.000
65	6	0	0	1.000	1.000	1.000
66	8	0	0	1.000	1.000	1.000
67	6	1	0	0.857	1.000	0.923
68	7	0	0	1.000	1.000	1.000
69	14	0	0	1.000	1.000	1.000
70	7	0	0	1.000	1.000	1.000
71	11	0	1	1.000	0.917	0.957
72	5	0	0	1.000	1.000	1.000
73	14	0	0	1.000	1.000	1.000
74	11	1	3	0.917	0.786	0.846
75	12	2	0	0.857	1.000	0.923
76	9	1	0	0.900	1.000	0.947
77	7	2	1	0.778	0.875	0.824
78	10	0	0	1.000	1.000	1.000
79	7	0	0	1.000	1.000	1.000
80	11	2	0	0.846	1.000	0.917
81	14	4	1	0.778	0.933	0.848
82	8	0	0	1.000	1.000	1.000
83	9	0	0	1.000	1.000	1.000
84	11	0	0	1.000	1.000	1.000
85	10	0	1	1.000	0.909	0.952
86	7	0	1	1.000	0.875	0.933
87	12	1	0	0.923	1.000	0.960
88	11	0	1	1.000	0.917	0.957
89	8	1	0	0.889	1.000	0.941
90	6	1	1	0.857	0.857	0.857
91	10	0	1	1.000	0.909	0.952
92	7	0	0	1.000	1.000	1.000
93	14	0	1	1.000	0.933	0.966
94	11	1	0	0.917	1.000	0.957
95	6	1	0	0.857	1.000	0.923

Metrici per-clasă pentru predicțiile făcute asupra datelor de validare



Statistici despre metriki determinate în funcție de predicții

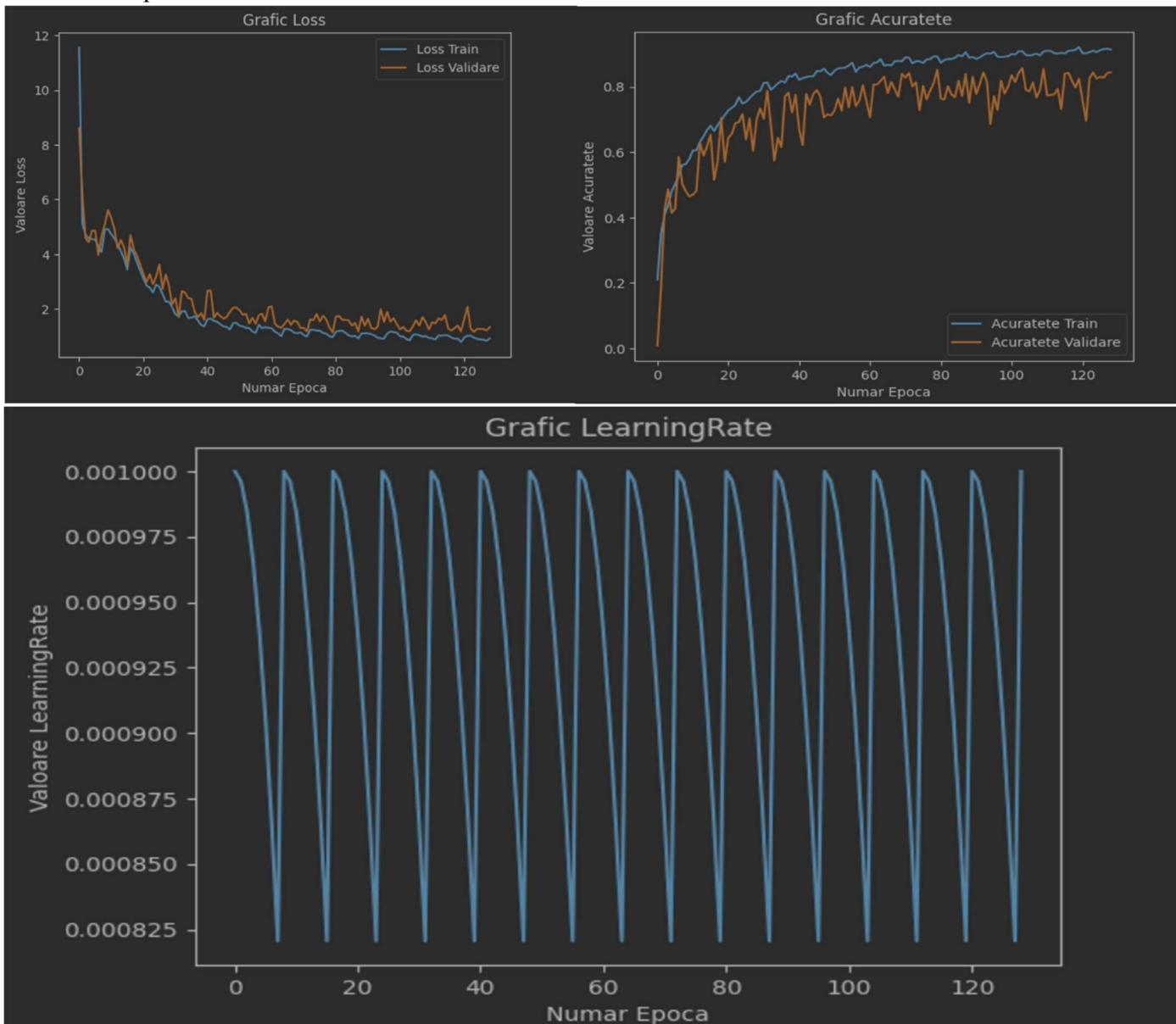
Hyperparameter tuning

LearningRate-uri testate

- Cosine Annealing Ciclic

```
''' callback pentru care definesc un scheduler al ratei de invatare care sa simuleze un Cosine Annealing Ciclic '''
callback_cosine_annealing_lr = callbacks.LearningRateScheduler(
    lambda ep: 0.00001 + (param_rata_initiala_lr - 0.00001) * ((math.cos((math.pi * (ep % int(param_epoci / 25)))) / 25) +
    1) / 2))
```

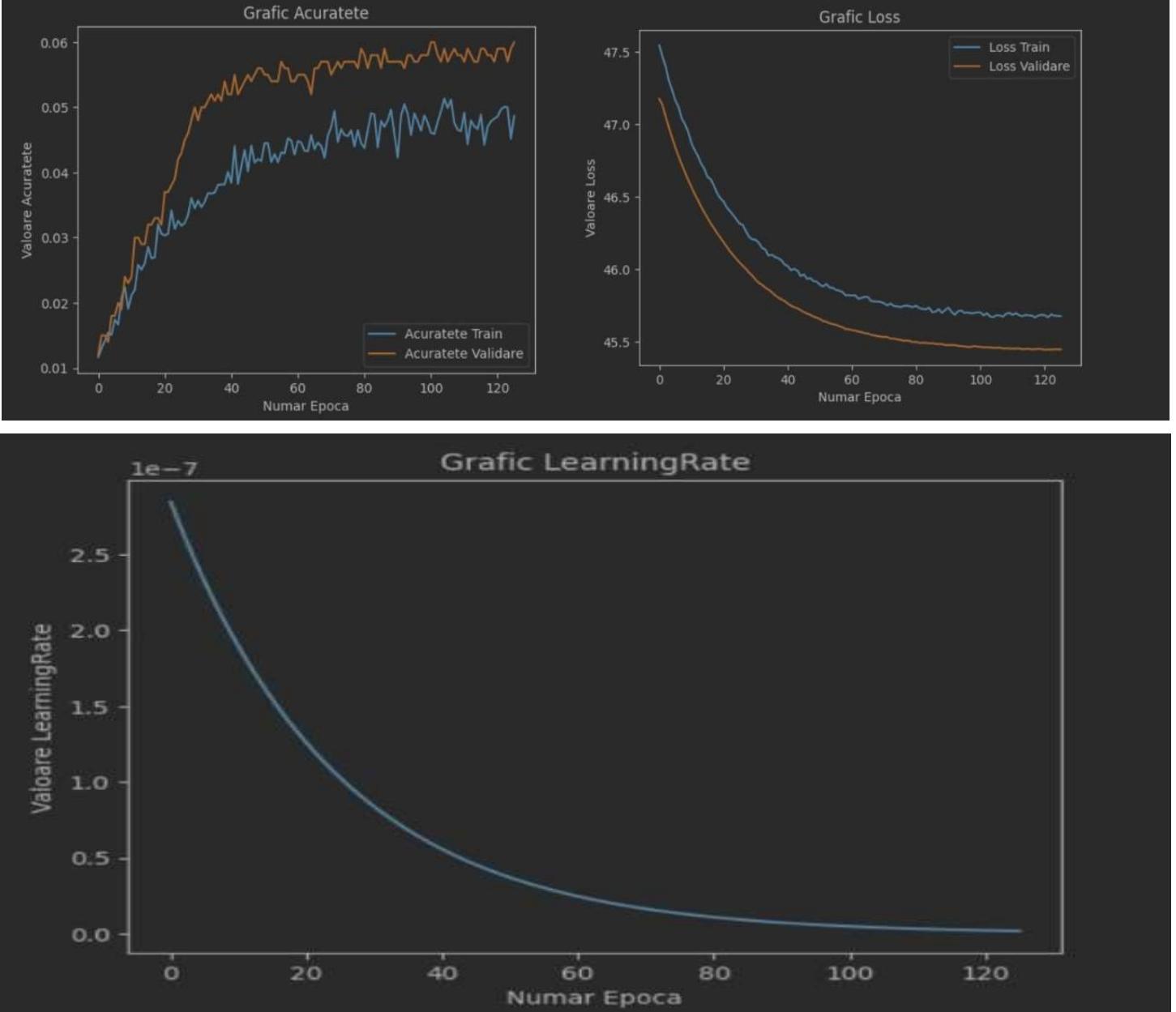
În cadrul acestei funcții setez o valoare minimă a learning rate-ului (0.00001) și un step size (25), iar pe parcursul antrenării, din 25 în 25 de epoci, modelul revine la learning rate-ul inițial și scade treptat.



Graficul asociat comportamentului ratei de învățare (cosine-ciclic)

- Exponential Decay

```
''' callback pentru care definesc un scheduler al ratei de invatare care sa simuleze un Exponential Decay LR '''
callback_exp_decay_lr = callbacks.LearningRateScheduler(
    lambda ep: param_rata_initiala_lr * param_rata_scdere ** (ep + param_epoci))
```



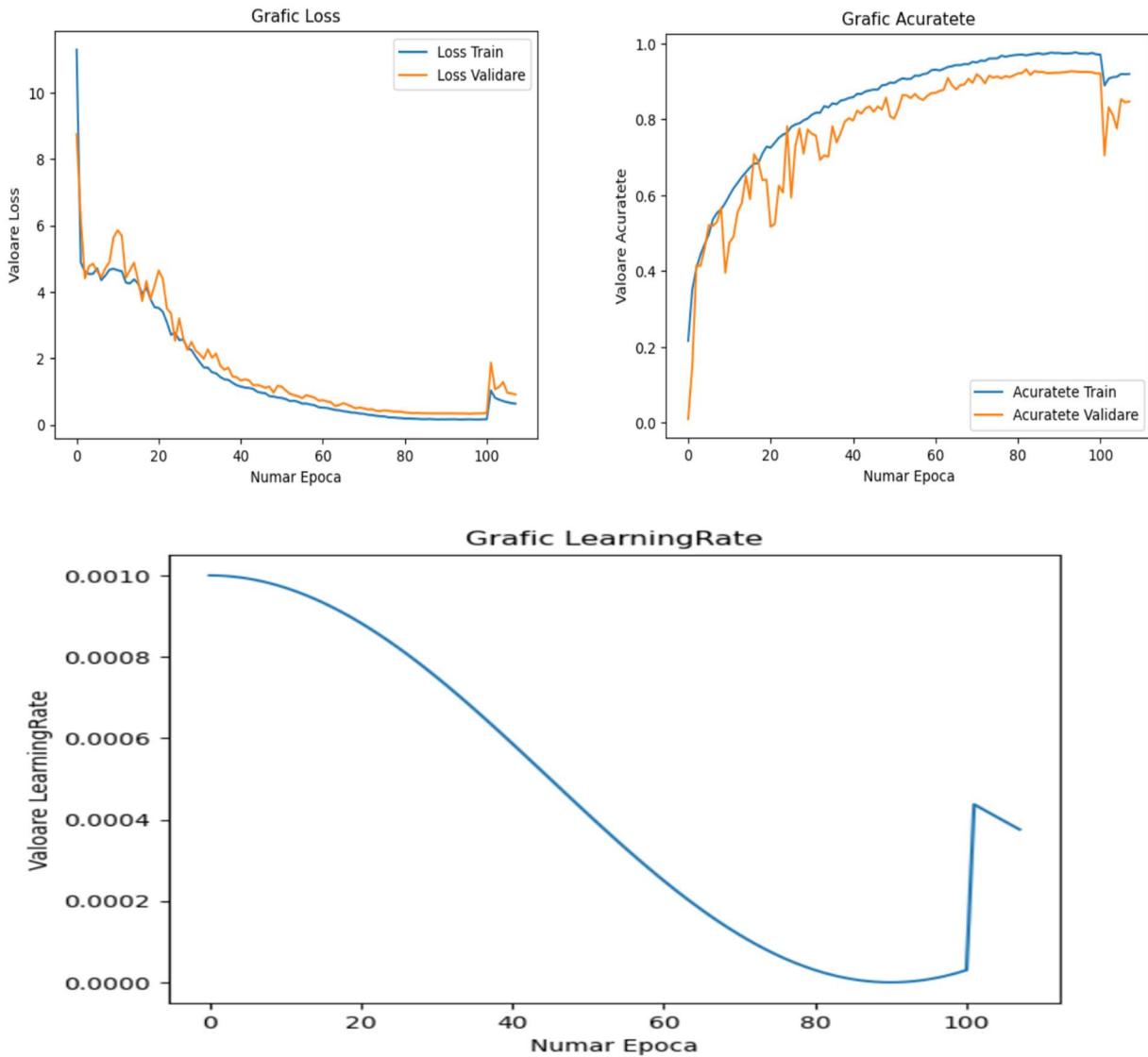
Graficul asociat comportamentului ratei de învățare (exponential-decay)

Putem observa faptul că modelul nu a evoluat deloc bine pentru această funcție asociată ratei de învățare, `param_rata_scdere` fiind egal cu 0.96, iar acuratetea maximă 0.06%.

- LearningRate bazat pe intuiție

```
callback_cosine_annealing_lr = callbacks.LearningRateScheduler(
    lambda ep: param_rata_initiala_lr * (1 + math.cos(
        math.pi * ep / (param_epoci - 110))) / 2 if ep <= 100 else param_rata_initiala_lr * (
        1 + cos(math.pi * (ep - 20) / (param_epoci - 50))) / 2)
```

Pentru acest comportament al ratei de învățare am testat mai multe de epoci, și am văzut ca în primele epoci învață mai mult dacă micșorez mai mult valoarea learning rate-ului, iar mai târziu trebuie adaptată, deoarece obțineam un learning rate care într-un punct creștea înapoi, deoarece simulam o rată de învățare circulară, fără să realizez acest lucru. Cea mai mare problemă constă în faptul că de câte ori modific numărul de epoci, după ce atingeam un scor bun, trebuie să adaptez learning rate-ul pentru a împărți la aceleași valori la care am văzut că modelul a funcționat optim.

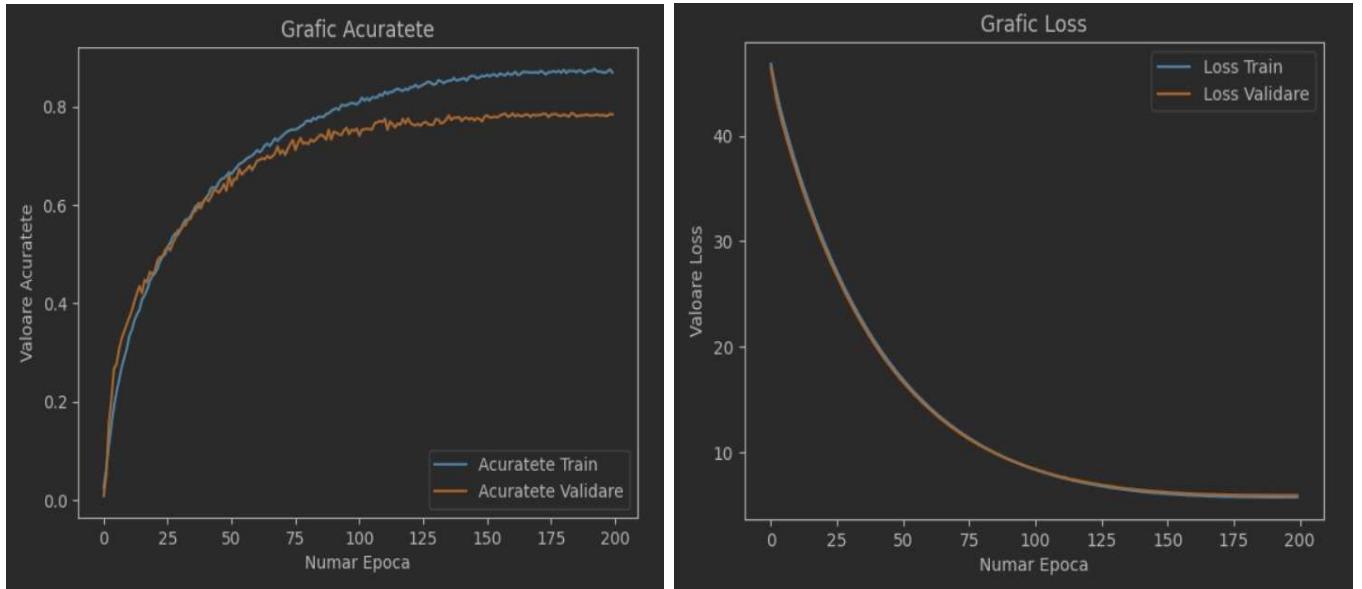


Graficul asociat comportamentului ratei de învățare (un fel de cosine warmUp)

Astfel, analizând graficele, putem observa faptul că, în epoca 100, acuratetea modelului crește, deoarece se schimba calculul ratei de învățare asociate.

Optimizatori testați

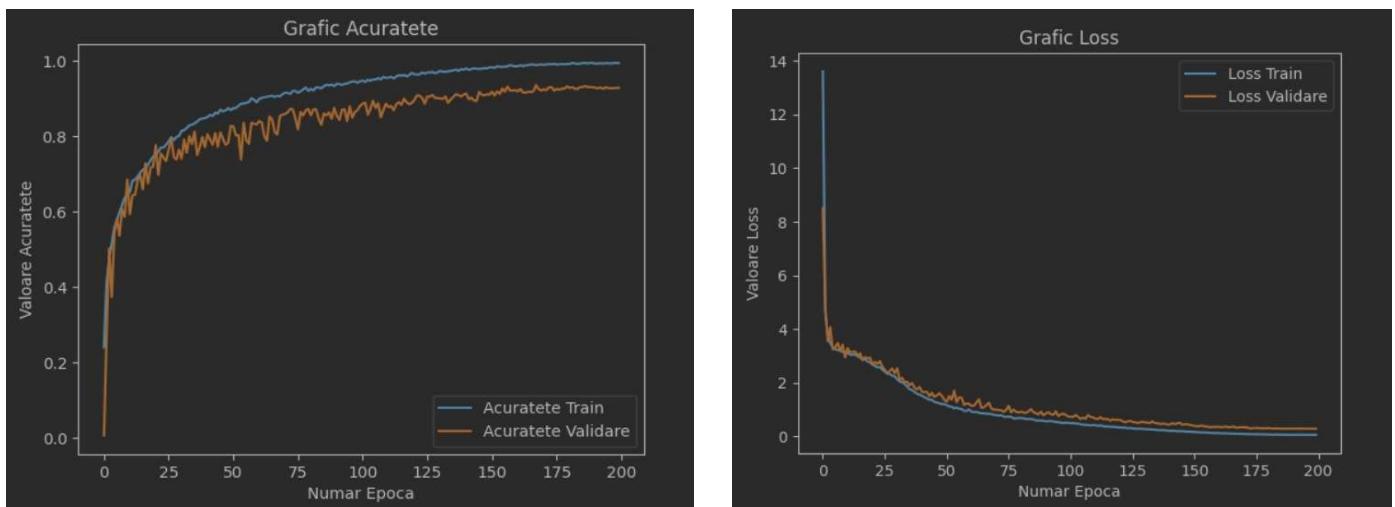
- SGD, cu weight-decay de 0.001



Evoluția acurateței și a loss-ului

Putem observa modul în care fluctuează acuratețea modelului și loss-ul în momentul în care folosim „Stochastic Gradient Descent” cu un weight-decay mic, și cum modelul pierde din performanță. Acest grafic este generat pentru aceeași arhitectură inclusă în temă.

- Adam, cu weight decay de 0.04



Evoluția acurateței și a loss-ului

Pentru aceeași arhitectură, am folosit un optimizator de tip Adam, cu un weight decay puțin mai mare, mai exact 0.04, dar pentru care modelul nu se adaptează la fel de bine. Obțin un scor mai mare în primele epoci, dar ulterior stagnează și nu mai are îmbunătățiri vizibile. De aceea, eu prefer o valoare mai mică pentru scăderea ponderilor.

Numărul de epoci

Pe parcursul întregii competiții, numărul de epoci a variat între 20 și 300 de epoci. Inițial, am crezut că este suficient să adaug un număr mic de epoci, dar modelul nu învăță prea mult. Ulterior, când am adăugat scheduler pentru rata de învățare, am descoperit că de cele mai ulte ori, modelul are o performanță mai bună în primele 50 de epoci, având creșteri foarte brusete. Putem observa mai jos, faptul că modelul a reușit să atingă o acuratețe de 91%, chiar în primele 50 de epoci.

Totuși, am ales să las ca hiperparametru un număr mai mare de epoci pentru antrenare, deoarece aceste „spike-uri” în acuratețe nu s-au dovedit neapărat bune. Acea submisie cu 91% în primele epoci a obținut un scor de 86% pe platformă, deoarece modelul cel mai probabil supraînvăță datele de test.

Hiperparametrii testați

Epoci	HIPERPARAMETRII						RESULTATE		
	Learning Rate Initial	Rata Scadere	Indulgenta	Dropout	Regularizare Kernel	Dimensiune Lot	Acuratete Validare	Acuratete Train	Epoci parcurse
100	0.001	0.5	25	0.15	0.035	48	92.2	98.4	100
100	0.001	0.1	25	0.15	0.035	48	90.4	99.37	100
100	0.001	0.3	25	0.4	0.035	48	92.8	98.09	100
100	0.001	0.3	10	0.4	0.001	64	93.4	99.56	100
100	0.01	0.3	10	0.25	0.04	64	88.01	92.74	100
200	0.001	0.5	10	0.3	0.03	16	80.8	78.2	53
200	0.002	0.1	15	0.2	0.025	32	88.9	94.48	97
50	0.001	0.5	15	0.25	0.025	94	91.9	97.4	50
50	0.001	0.9	15	0.25	0.025	94	91.7	96.6	50
100	0.001	1.5	15	0.25	0.025	94	92.4	97.8	100
100	0.001	2.5	15	0.4	0.025	94	91.8	96.01	100

Hiperparametrii testați pentru arhitectura modelului final

Acest tabel conține hiperparametrii testați pentru arhitectura modelului încărcat în cadrul temei și am surprins acuratețea finală a modelului, dar și momentul în care este oprit dacă nu prezintă o evoluție într-un anumit număr de epoci. De aici, am observat faptul că o dimensiune mai mare a lotului este utilă și obține valori mai bune decât dimensiunile mici.

Alte arhitecturi de CNN încercate

Spre finalul competiției, am încercat să îmbunătățesc rețeaua adăugând zgomot Gaussian, asupra imaginilor, însă nu am obținut niciun rezultat pozitiv. După mai multe încercări, am reușit să implementez o rețea care să folosească „skip layers”. Astfel, am concatenat convoluția doi la convoluția șase, înainte de a aplica ultimul dropout. Făcând acest lucru, reușesc să preiau caracteristici învățate de către model în stadiile inițiale și să le folosesc mai târziu. Deci, prin concatenarea straturilor convoluționale reușesc să am mai multe informații folositoare, când mă apropiez de straturile dense.

Pentru a putea adăuga acest strat concatenat, am modificat arhitectura modelului și am renunțat la implementarea acestuia prin folosirea „Sequential”, care înfățoară straturile modelului, și am implementat un model din clasa „Model”. Pentru aceasta, am inițializat un constructor în care să includ straturile necesare, și le-am unificat în cadrul metodei „call”, care este metodă standard a clasei „Model”, pe care eu am suprascris-o. Metoda „call” primește ca parametru inputurile, o variabilă booleană care indică diferite comportamente ale modelului în funcție de etapă (antrenare sau realizare de predicții). Prin apelul metodei „fit” sau „compile”, variabila „training” capătă implicit valoarea „true”.

Pentru a putea concatena cele două straturi, a trebuit să aplic un down-sampling, reducând dimensiunea celui de-al doilea strat convecțional prin trecerea acestuia printr-un strat de MaxPooling2D cu o dimensiune de (8x8).

Acest model testat, conține aceleași straturi ca cel predat, și aceleași callback-uri definite (checkpoint, early stopping și learning rate scheduler).

Totuși, nu îl consider un model reușit, deoarece are tendința de a supraînvăța datele de test, iar pe platformă, diferența dintre valoarea prezisă pe local și punctajul obținut pentru submisie este destul de mare. De asemenea, salvarea modelului nu mai poate fi făcută într-o filă cu extensia „.h5”, ci trebuie salvat sub formatul „tf”, ceea ce întoarce un warning care mă avertizează că nu au putut fi salvate toate caracteristicile modelului, și nu pot fi refolosite.

```
def __init__(self, param_rata_dropout=0.25, param_regularizare_kernel=0.03,
param_numar_clase=96):
    super(Structura_Model_CNN, self).__init__()

    self.conv_1 = layers.Conv2D(32, (3, 3), strides=1, padding='same',
activation='relu')
    self.batch_norm_1 = layers.BatchNormalization()

    self.conv_2 = layers.Conv2D(64, (3, 3), strides=1, padding='same',
activation='relu')
    self.batch_norm_2 = layers.BatchNormalization()
    self.max_pool_1 = layers.MaxPooling2D(pool_size=(2, 2))

    self.conv_3 = layers.Conv2D(64, (3, 3), strides=1, padding='same',
activation='relu')
    self.batch_norm_3 = layers.BatchNormalization()

    self.conv_4 = layers.Conv2D(128, (3, 3), strides=1, padding='same',
activation='relu')
    self.batch_norm_4 = layers.BatchNormalization()
    self.max_pool_2 = layers.MaxPooling2D(pool_size=(2, 2))

    self.drop_out_1 = layers.Dropout(rate=param_rata_dropout)

    self.conv_5 = layers.Conv2D(256, (3, 3), strides=1, padding='same',
activation='relu')
    self.batch_norm_5 = layers.BatchNormalization()

    self.conv_6 = layers.Conv2D(512, (3, 3), strides=1, padding='same',
activation='relu')
    self.batch_norm_6 = layers.BatchNormalization()
    self.max_pool_3 = layers.MaxPooling2D(pool_size=(2, 2))

    self.drop_out_2 = layers.Dropout(rate=param_rata_dropout)

    self.flatten = layers.Flatten()

    self.dense_1 = layers.Dense(512, activation='relu',
kernel_regularizer=regularizers.l2(param_regularizare_kernel))
    self.batch_norm_7 = layers.BatchNormalization()

    self.dense_2 = layers.Dense(128, activation='relu',
kernel_regularizer=regularizers.l2(param_regularizare_kernel))
    self.batch_norm_8 = layers.BatchNormalization()

    self.dense_3 = layers.Dense(param_numar_clase, activation='softmax')

    self.down_sampling = layers.MaxPooling2D(pool_size=(8, 8))
```

```
def call(self, inputs, training=False, mask=None):
    x = self.conv_1(inputs)
    x = self.batch_norm_1(x, training=training)

    x = self.conv_2(x)
    skip_layer = x
    x = self.batch_norm_2(x, training=training)
    x = self.max_pool_1(x)

    x = self.conv_3(x)
    x = self.batch_norm_3(x, training=training)

    x = self.conv_4(x)
    x = self.batch_norm_4(x, training=training)
    x = self.max_pool_2(x)

    x = self.drop_out_1(x, training=training)

    x = self.conv_5(x)
    x = self.batch_norm_5(x, training=training)

    x = self.conv_6(x)

    x = self.batch_norm_6(x, training=training)
    x = self.max_pool_3(x)

    skip_layer = self.down_sampling(skip_layer)
    x = layers.concatenate([skip_layer, x])

    x = self.drop_out_2(x, training=training)

    x = self.flatten(x)

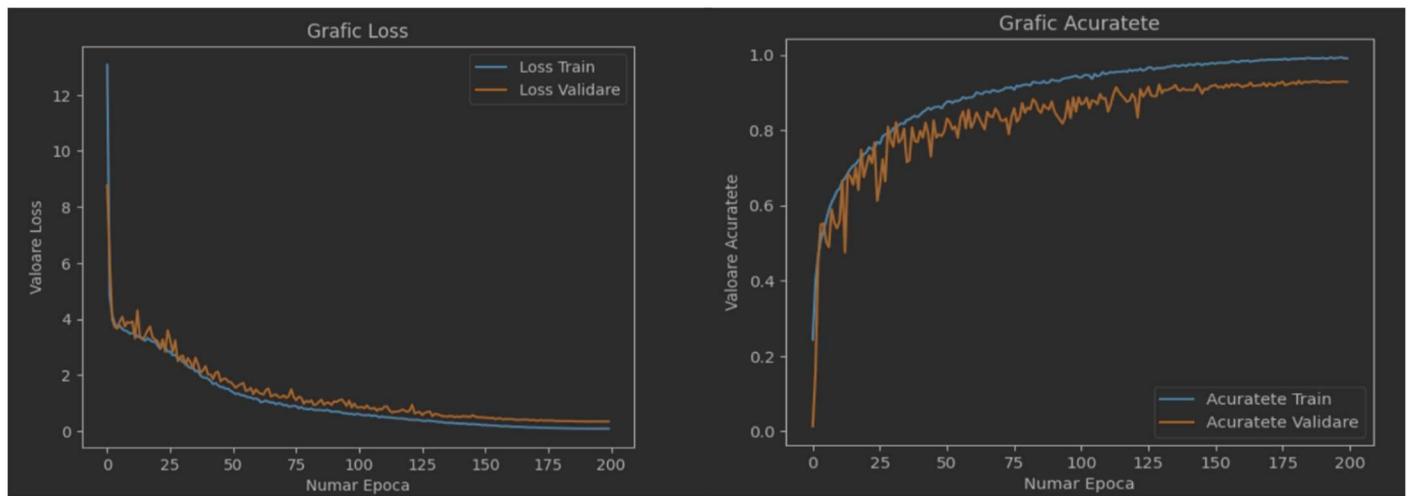
    x = self.dense_1(x)
    x = self.batch_norm_7(x, training=training)

    x = self.dense_2(x)
    x = self.batch_norm_8(x, training=training)

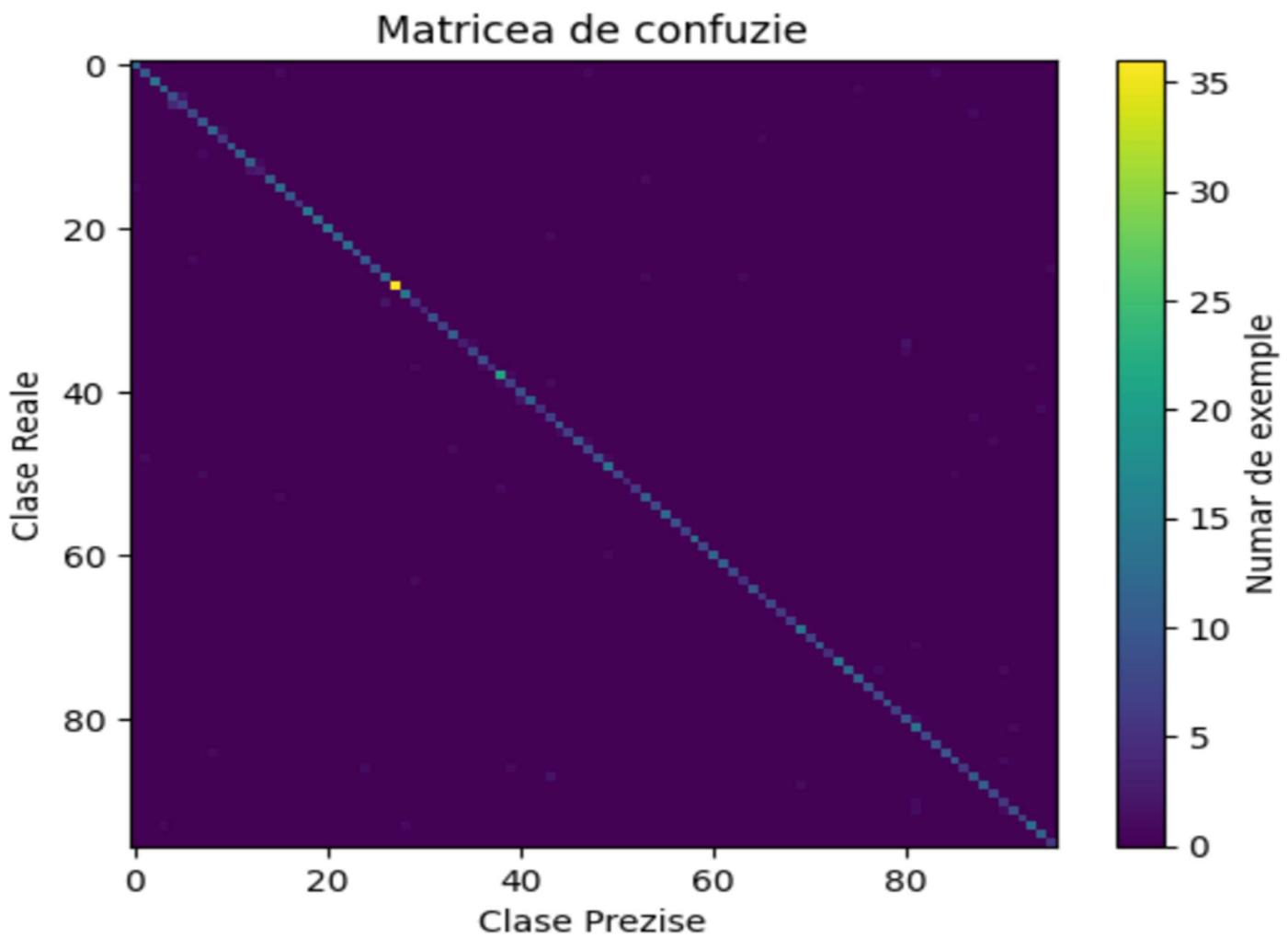
    outputuri = self.dense_3(x)

    return outputuri
```

Arhitectura modelului testat care să permită skip-layers



Grafcile asociate pentru acurate și loss

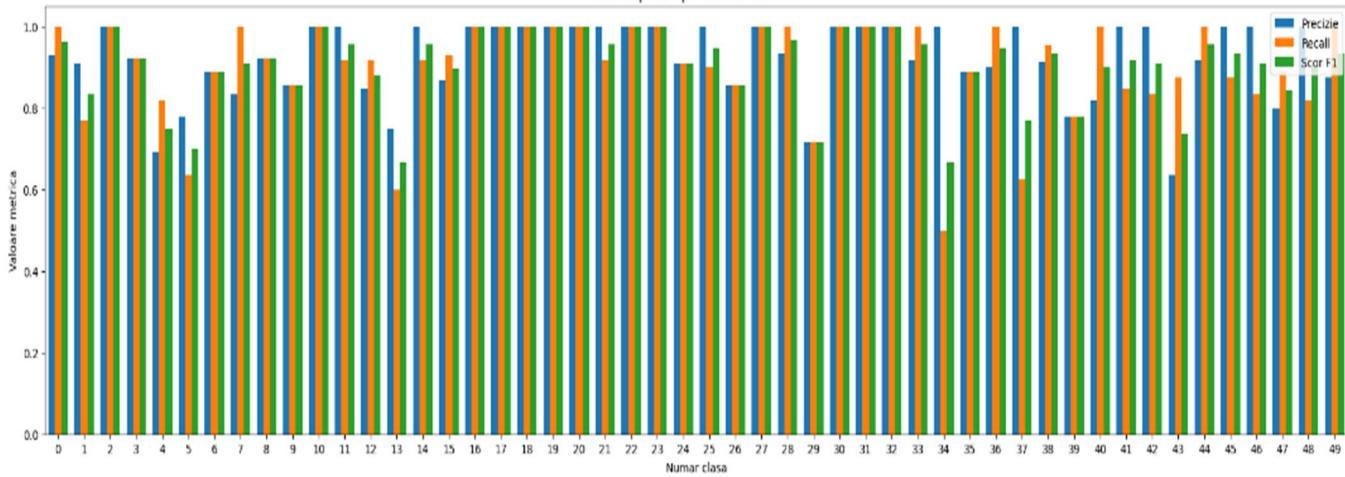


Acuratete totală: 93.1000000000001%

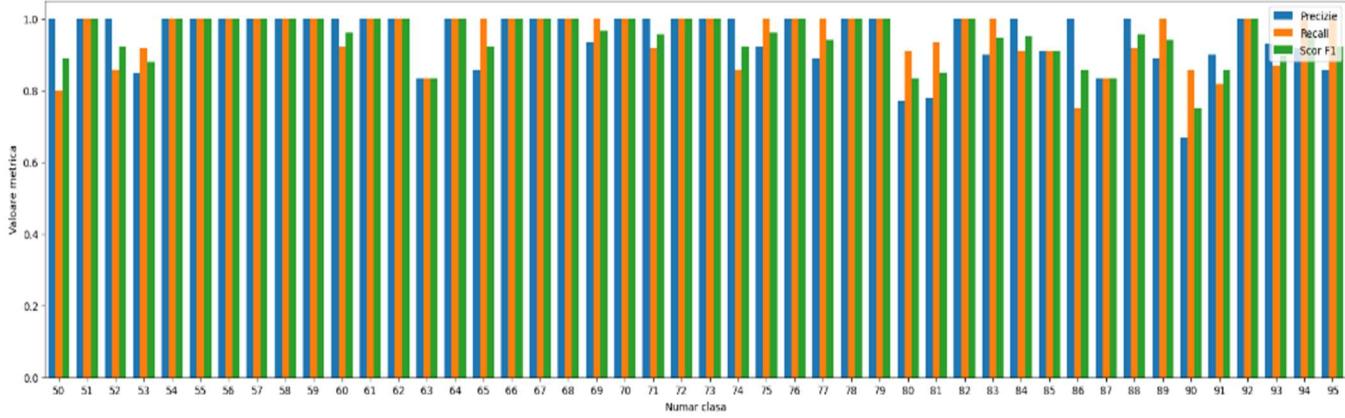
Statistică pentru fiecare clasa:

Clasa	Adevarat Pozitive	Fals Pozitive	Fals Negative	Precizie	Recall	F1
0	13	1	0	0.929	1.000	0.963
1	10	1	3	0.909	0.769	0.833
2	14	0	0	1.000	1.000	1.000
3	12	1	1	0.923	0.923	0.923
4	9	4	2	0.692	0.818	0.750
5	7	2	4	0.778	0.636	0.700
6	8	1	1	0.889	0.889	0.889
7	10	2	0	0.833	1.000	0.909
8	12	1	1	0.923	0.923	0.923
9	6	1	1	0.857	0.857	0.857
10	11	0	0	1.000	1.000	1.000
11	11	0	1	1.000	0.917	0.957
12	11	2	1	0.846	0.917	0.880
13	3	1	2	0.750	0.600	0.667
14	11	0	1	1.000	0.917	0.957
15	13	2	1	0.867	0.929	0.897
16	10	0	0	1.000	1.000	1.000
17	6	0	0	1.000	1.000	1.000
18	15	0	0	1.000	1.000	1.000

Metrici pentru primele 50 de clase



Metrici pentru restul claselor



Statistică despre metricile determinate în funcție de predicții

Observație importantă

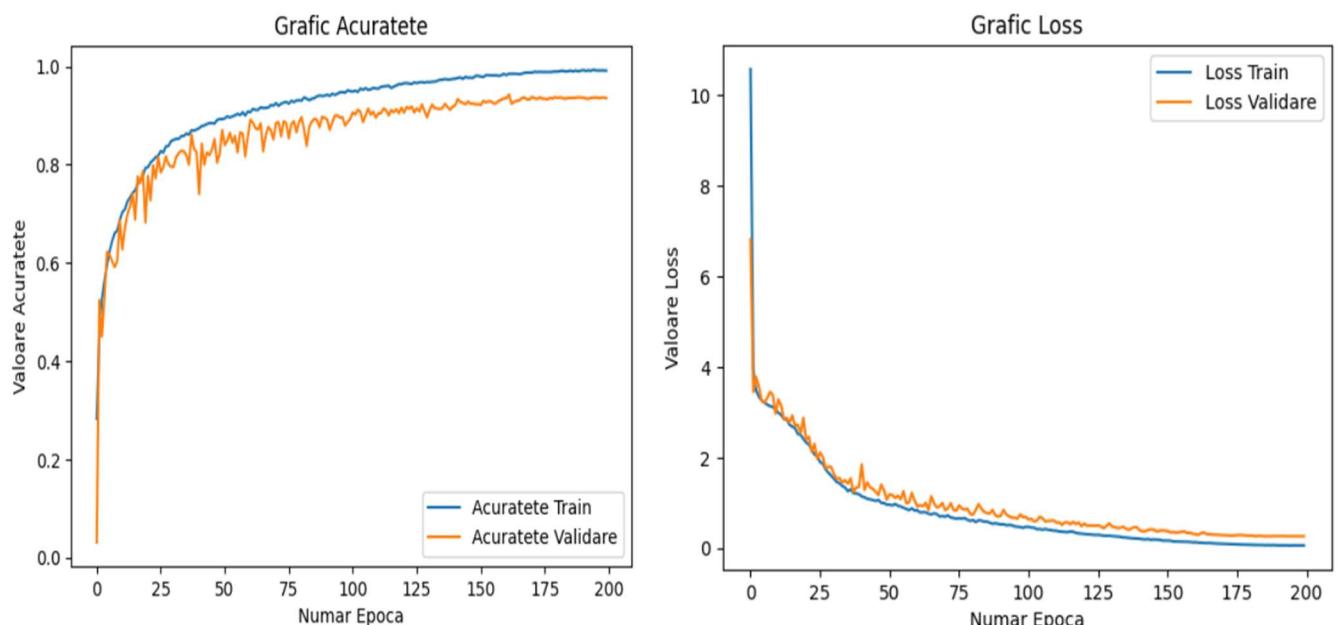
Deoarece am folosit pentru augmentarea imaginilor o funcție care să ștergă dreptunghiuri din acestea într-un mod randomizat, pentru aceeași hiperparametru și aceeași arhitectură a modelului, în cadrul mai multor rulări a programului, modelul are capacitatea de a face predicții care diferă destul de mult din punct de vedere al acurateței. Observăm că avem cazul în care dimensiunile determinante pentru ștergere depășesc dimensiunile imaginii propriu zise (de exemplu, pentru $\text{sqrt}(0.4 \cdot 64 \cdot 64) \cdot 3$ obținem valoarea 70, care este mai mare decât înălțimea imaginii), iar în acel caz, atrbuim lungimii sau lățimii întreaga dimensiune care corespunde imaginii. Din aceste motive, am constatat faptul că, pentru modelul încărcat la predare, am obținut scoruri cuprinse între 90.8% și 94.1%.

Mențiune

După anularea primei competiții, în cadrul căreia a fost remarcat faptul că ultimele două cifre din denumirea imaginii corespund clasei acesteia, am preluat cele 7000 de imagini de test și le-am folosit în antrenarea modelului final.

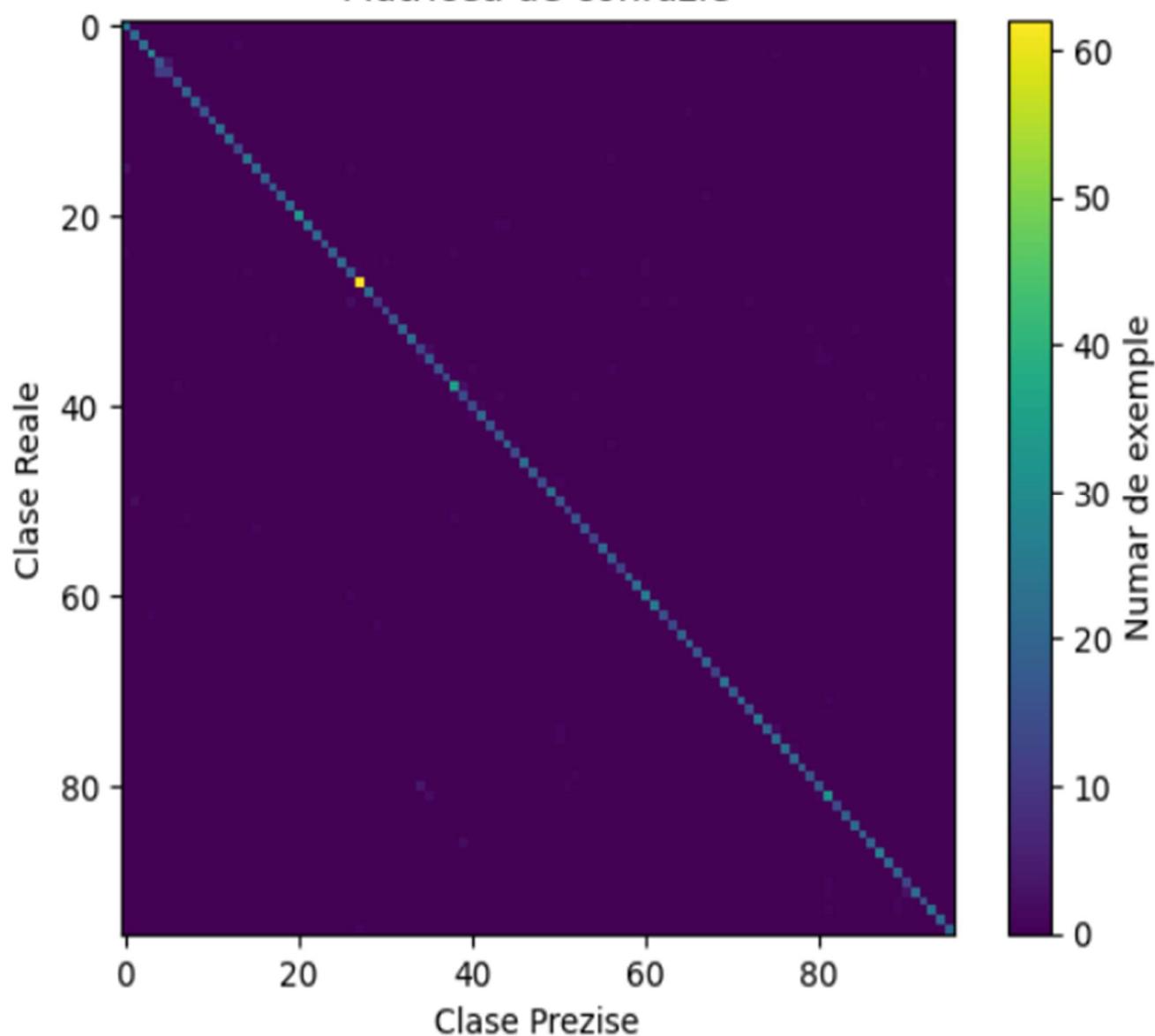
Astfel, am atribuit pentru antrenare 6000 de imagini în plus, iar pentru validare restul de 1000, iar ca etichete corecte le-am atribuit ultimele două cifre din nume. Chiar dacă știu că nu sunt 100% corecte aceste atribuiriri, o submisie care conținea doar ultimele două cifre obține un scor aproximativ egal cu 98%, aşadar rata de eroare este destul de mică.

Având la dispoziție mai multe date pentru antrenare, am reușit să îmbunătățesc performanța modelului, obținând ușor scoruri ale acurateței care să depășească 94%. Mai jos voi prezenta mai multe grafice și imagini care indică comportamentul modelului pe care l-am selectat pentru predare, în cadrul unei antrenări în care folosește datele „extra”.



Grafice ce prezintă evoluția acurateței și a funcției de loss

Matricea de confuzie



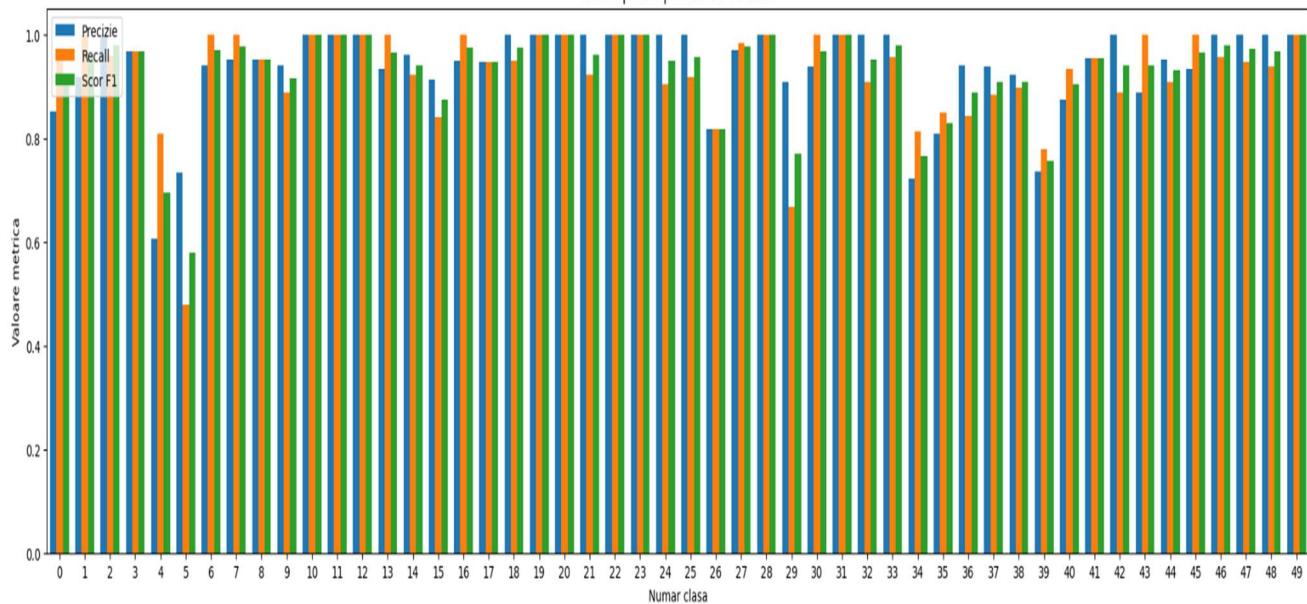
Matricea de confuzie asociată predicțiilor făcute pentru datele de validare

Acuratețe totală: 94.3%

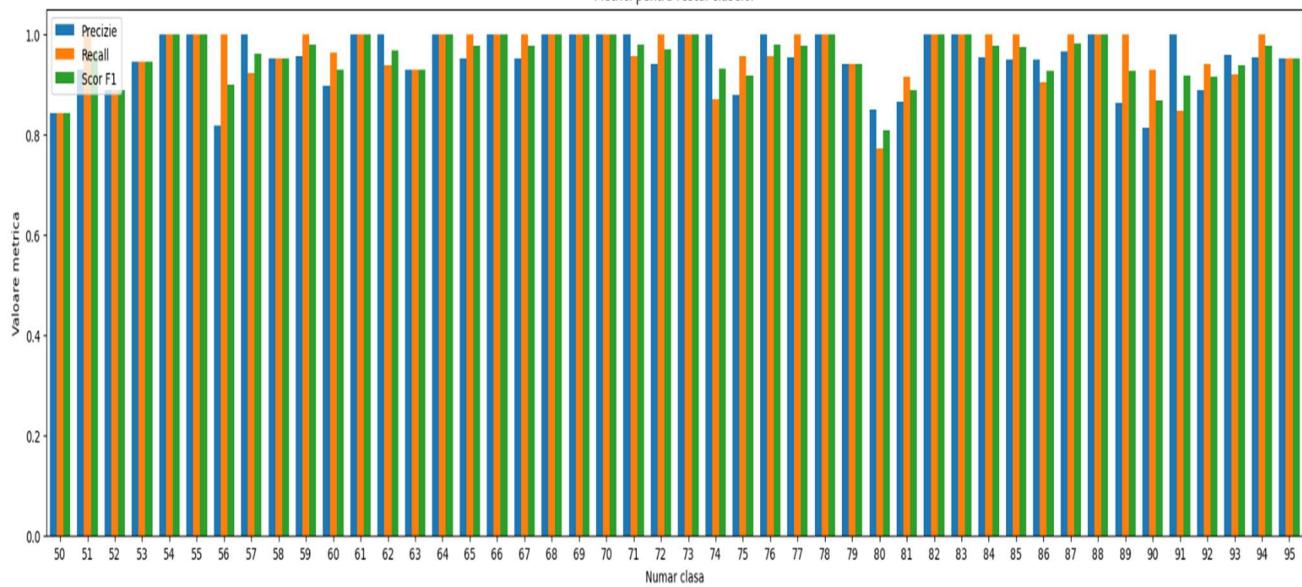
Statistică pentru fiecare clasa:

Clasa	Adevarat Pozitive	Fals Pozitive	Fals Negativ	Precizie	Recall	F1	52	16	2	2	0.889	0.889	0.889
0	23	4	1	0.852	0.958	0.902	54	12	0	0	1.000	1.000	1.000
1	22	2	0	0.917	1.000	0.957	55	21	0	0	1.000	1.000	1.000
2	23	0	1	1.000	0.958	0.979	56	18	4	0	0.818	1.000	0.900
3	30	1	1	0.968	0.968	0.968	57	12	0	1	1.000	0.923	0.960
4	17	11	4	0.607	0.810	0.694	58	20	1	1	0.952	0.952	0.952
5	11	4	12	0.733	0.478	0.579	59	22	1	0	0.957	1.000	0.978
6	16	1	0	0.941	1.000	0.970	60	26	3	1	0.897	0.963	0.929
7	20	1	0	0.952	1.000	0.976	61	25	0	0	1.000	1.000	1.000
8	20	1	1	0.952	0.952	0.952	62	15	0	1	1.000	0.938	0.968
9	16	1	2	0.941	0.889	0.914	63	13	1	1	0.929	0.929	0.929
10	19	0	0	1.000	1.000	1.000	64	19	0	0	1.000	1.000	1.000
11	23	0	0	1.000	1.000	1.000	65	20	1	0	0.952	1.000	0.976
12	22	0	0	1.000	1.000	1.000	66	16	0	0	1.000	1.000	1.000
13	14	1	0	0.933	1.000	0.966	67	20	1	0	0.952	1.000	0.976
14	24	1	2	0.960	0.923	0.941	68	12	0	0	1.000	1.000	1.000
15	21	2	4	0.913	0.840	0.875	69	24	0	0	1.000	1.000	1.000
16	19	1	0	0.950	1.000	0.974	70	17	0	0	1.000	1.000	1.000
17	18	1	1	0.947	0.947	0.947	71	22	0	1	1.000	0.957	0.978
18	19	0	1	1.000	0.950	0.974	72	16	1	0	0.941	1.000	0.970
19	21	0	0	1.000	1.000	1.000	73	25	0	0	1.000	1.000	1.000
20	33	0	0	1.000	1.000	1.000	74	20	0	3	1.000	0.870	0.930
21	24	0	2	1.000	0.923	0.960	75	22	3	1	0.880	0.957	0.917
22	20	0	0	1.000	1.000	1.000	76	22	0	1	1.000	0.957	0.978
23	18	0	0	1.000	1.000	1.000	77	21	1	0	0.955	1.000	0.977
24	19	0	2	1.000	0.905	0.950	78	18	0	0	1.000	1.000	1.000
25	22	0	2	1.000	0.917	0.957	79	16	1	1	0.941	0.941	0.941
26	18	4	4	0.818	0.818	0.818	80	17	3	5	0.850	0.773	0.810
27	62	2	1	0.969	0.984	0.976	81	32	5	3	0.865	0.914	0.889
28	22	0	0	1.000	1.000	1.000	82	14	0	0	1.000	1.000	1.000
29	10	1	5	0.909	0.667	0.769	83	18	0	0	1.000	1.000	1.000
30	15	1	0	0.938	1.000	0.968	84	21	1	0	0.955	1.000	0.977
31	16	0	0	1.000	1.000	1.000	85	19	1	0	0.950	1.000	0.974
32	20	0	2	1.000	0.909	0.952	86	19	1	2	0.950	0.905	0.927
33	22	0	1	1.000	0.957	0.978	87	27	1	0	0.964	1.000	0.982
34	13	5	3	0.722	0.812	0.765	88	21	0	0	1.000	1.000	1.000
35	17	4	3	0.810	0.850	0.829	89	19	3	0	0.864	1.000	0.927
36	16	1	3	0.941	0.842	0.889	90	13	3	1	0.812	0.929	0.867
37	15	1	2	0.938	0.882	0.909	91	22	0	4	1.000	0.846	0.917
38	35	3	4	0.921	0.897	0.909	92	16	2	1	0.889	0.941	0.914
39	14	5	4	0.737	0.778	0.757	93	23	1	2	0.958	0.920	0.939
40	14	2	1	0.875	0.933	0.903	94	21	1	0	0.955	1.000	0.977
41	21	1	1	0.955	0.955	0.955	95	20	1	1	0.952	0.952	0.952
42	16	0	2	1.000	0.889	0.941							
43	16	2	0	0.889	1.000	0.941							
44	20	1	2	0.952	0.909	0.938							
45	14	1	0	0.933	1.000	0.966							
46	22	0	1	1.000	0.957	0.978							
47	18	0	1	1.000	0.947	0.973							
48	15	0	1	1.000	0.938	0.968							
49	23	0	0	1.000	1.000	1.000							
50	16	3	3	0.842	0.842	0.842							
51	13	1	0	0.929	1.000	0.963							

Metrici pentru primele 50 de clase



Metrici pentru restul claselor



Statistici despre metricile determinate în funcție de predicții

Antrenând modelul pe acest nou set de date (care nu a inclus date din afara competiției) am reușit să reduc din diferența dintre rezultatele referitoare la acuratețe pe care le obțineam pentru datele de validare, și rezultatul întors de platformă în momentul în care adăugam o submisie.

Concluzii

Ca model final pentru K-NN am ales numărul de vecini să fie 10, iar distanța calculată de tip „cityblock”.

Ca model final pentru CNN am implementat un model cu șase straturi convolutionale și trei straturi dense (fully connected), care conțin minim 32 și maxim 512 filtre. Am inclus și două straturi de dropout, unul după primele patru straturi convolutionale, iar altul înainte de a transforma inputul multidimensional într-unul unidimensional, adică înainte de a conecta straturile dense. Ca și optimizator am ales Adam, cu o scădere a ponderilor mică (de 0.0001), astfel încât să îmbunătățesc performanța. Funcția de activare folosită predominant este „relu”, singurul strat care utilizează o altă funcție fiind ultimus strat dens, unde se realizează clasificarea și am adăugat funcția „softmax”, care să realizeze o distribuție a probabilităților pentru includerea într-o dintre 96 de clase. De asemenea, sunt prezente straturi de max pooling, cu o dimensiune de (2x2), care să reducă dimensiunile inputului.

Modelul conține mai multe callback-uri. Unul dintre ele este un checkpoint, folosit pentru a garanta salvarea celei mai bune structuri obținute pe parcursul antrenării și generarea predicțiilor pe baza acestuia. Un alt callback este early stopping, care să realizeze oprirea antrenării modelului în momentul în care acesta nu își îmbunătățește performanța într-un număr dat de epoci. Iar ultimul este un scheduler pentru rata de învățare, astfel încât să o adapteze în funcție de epoca actuală.

Pentru procesarea datelor, redimensionez și normalizez imaginile și transform etichetele citite din șiruri de caractere în valori întregi, astfel încât să pot aplica funcția de loss asupra acestora. Pentru Augmentarea imaginilor, aplic întoarceri orizontale și verticale ale acestora, iar suplimentar am definit o funcție care să steargă într-un mod randomizat dreptunghiuri din pozele inițiale.