

Universitatea din Bucureşti
Facultatea de Matematică și Informatică

Documentație Deep Hallucination Classification (Kaggle)

Proiect machine learning

Student: Ghinea Dragoș Dumitru
Specializarea: Informatică
Grupa: 232
Anul II

Profesor Coordonator: Prof.dr. Radu Ionescu
Profesor Laborator: Asist.(det.) drd. Vlad Hondru

Bucureşti
2023

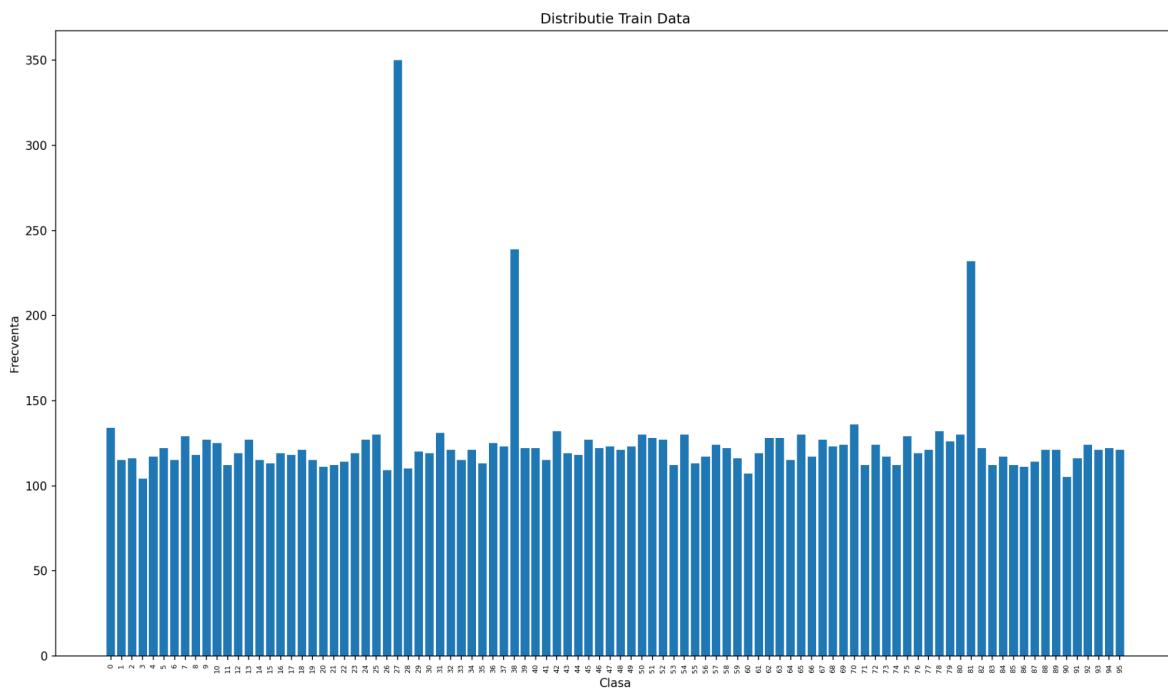
Cuprins

- 1. Setul de date**
- 2. Support Vector Machine (SVM)**
 - 2.1. Raw Pixels Representation (Chosen Feature Set)
 - 2.2. Histogram of Oriented Gradients (Chosen Feature Set)
 - 2.3 Concluzie SVM
- 3. Convolutional Neural Network (CNN)**
 - 3.1. Introducem normalizare
 - 3.2. Comparație Adam, SGD, RMSprop
 - 3.3. Weight decay
 - 3.4. Data Augmentation
 - 3.5. Learning Rate & Learning Rate Scheduler
 - 3.6. Model Upgrade
 - 3.7. Activation Function
 - 3.8. Batch Normalization
 - 3.9. Model upgrade 2
 - 3.10. Revine RandomAffine
 - 3.11. Mai multe epoci
 - 3.12. Early Stopping
 - 3.13. Oportunitate - Folosirea datelor de testare de la competiția anulată
 - 3.14. Skip Layer/Connection
- 4. Alte încercări pe CNN**
 - 4.1. Modificare Batch Size
 - 4.2. Dropout
 - 4.3. AdaptiveMaxPool vs AdaptiveAvgPool
 - 4.4. RandomInvert
 - 4.5. HOG Features
 - 4.6. Experimente pe transformări
- 5. Concluzie CNN**

➤ Setul de date:

Avem la dispoziție 12000 imagini de 64x64 de pixeli, RGB care trebuie clasificate în 96 de categorii.

Verificăm cum sunt distribuite și obținem diagrama de mai jos în care putem observa că majoritatea claselor sunt distribuite egal, cu excepția a trei dintre acestea:



➤ Support Vector Machine (SVM)

Un prim model ales este SVM, prin implementarea sa, Support Vector Classifier (SVC). În cod, funcția care execută SVM primește 3 parametrii: C, kernel, gamma.

➤ Raw Pixels Representation (Chosen Feature Set)

Pornim cu valorile default ($C=1$, $\text{kernel}=\text{'rbf'}$, $\text{gamma}=\text{'auto'}$) și începem să rulăm. Fără normalizare obținem o acuratețe foarte mică (<0.1), aplicăm normalizarea

standard și obținem o acuratețe de 0.389! În final, tot modificând acest parametru, C va ajunge să fie 10, aşa cum reiese din tabelul de mai jos:

c	acuratețe
0.1	0.143
1	0.389
10	0.405
25	0.403
50	0.403
80	0.403
100	0.403
1000	0.403
10000	0.403

Observăm că de la 25 în sus hiperplanul are o margine atât de mică încât nu mai învață, motiv pentru care și dacă îl creștem, de aici nu mai putem crește acuratețea (overfitting). Pe măsură ce C scade, modelul devine mai permisiv în clasificarea greșită a datelor, motiv pentru care nu reușește să învețe cum trebuie (underfitting).

Alegerea kernel-ului:

kernel	acuratețe
rbf	0.405
linear	0.289
poly	0.264

➤ Histogram of Oriented Gradients (Chosen Feature Set)

Urmează să încerc un alt tip de feature set, mai exact HOG (Histogram of Oriented Gradients).

Atunci când modelul este antrenat cu **HOG** (configurația: orientations = 9, pixels_per_cell = (8,8), cells_per_block=(2, 2), channel_axis=-1) obținem o acuratețe de 0.499!

Am eliminat normalizarea standard pentru a vedea cum se comportă modelul cu noul feature set, dar acuratețea a scăzut la 0.369, deci normalizarea se păstrează. Experimente pe configurația HOG:

orientations	pixels_per_cell	acuratețe
9	(8,8)	0.499
11	(8,8)	0.495
7	(8,8)	0.477
11	(5,5)	0.445

Creșterea numărului de orientări, scăderea numărului de pixeli per cell ar trebui să îmbunătățească captarea detaliilor, dar în cazul de față cea inițială este cea mai bună.

De menționat este și că, pe lângă faptul că HOG obține o acuratețe mai mare, rulează și mult mai rapid, având un număr de dimensiuni spațiale redus (raw pixels = 12288, hog = 1764).

Încercăm C diferite pe hog și observăm că diferă față de raw pixels representation:

C	acuratețe
1	0.482
10	0.499
30	0.5
60	0.5
300	0.5

Observăm o foarte mică îmbunătățire dacă setăm C puțin mai mare decât 10.

Dacă tot le-am testat individual, urmează să încerc o combinație de **raw pixels representation + HOG** pe configurația C=10, kernel='rbf', gamma='auto'. Facem acest lucru prin concatenarea feature-urilor hog, la cele de raw pixels representation, obținând astfel un set mai mare de feature-uri per imagine. Obținem o nouă acuratețe de 0.544!

Încercăm alegerea kernel-ului din nou:

kernel	acuratețe
rbf	0.544
linear	0.52
poly	0.426

Alte mențiuni:

* Am încercat să țin cont de cele 3 clase care apar mai frecvent, adăugând

```
class_weights = class_weight.compute_class_weight('balanced',
classes=np.unique(labeluri_validare), y=labeluri_validare)

svm_model = svm.SVC(C=C_var, kernel=kernel_var, gamma=gamma_var,
class_weight=dict(enumerate(class_weights)))
```

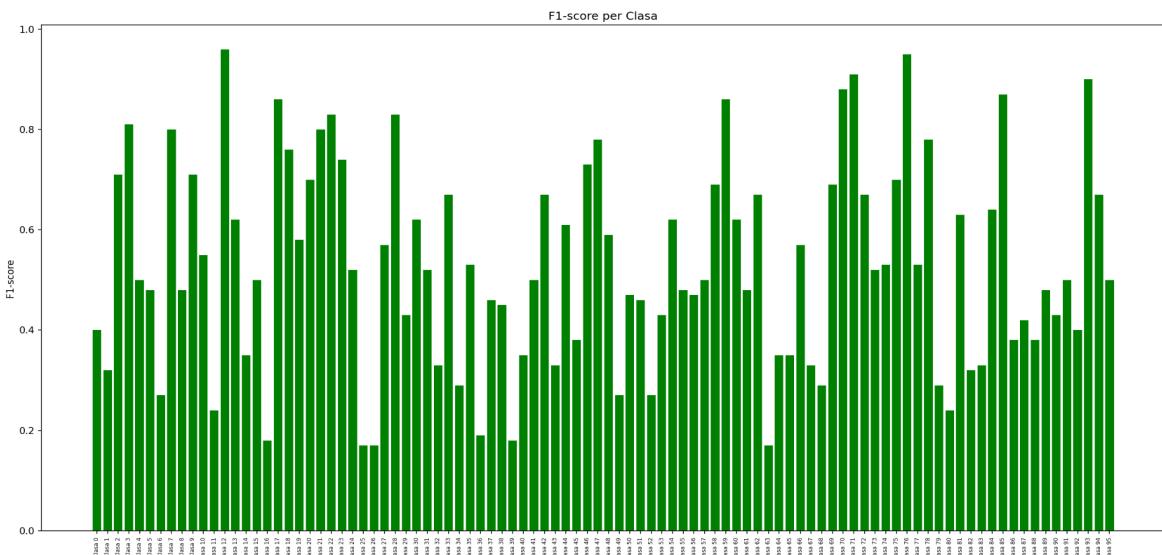
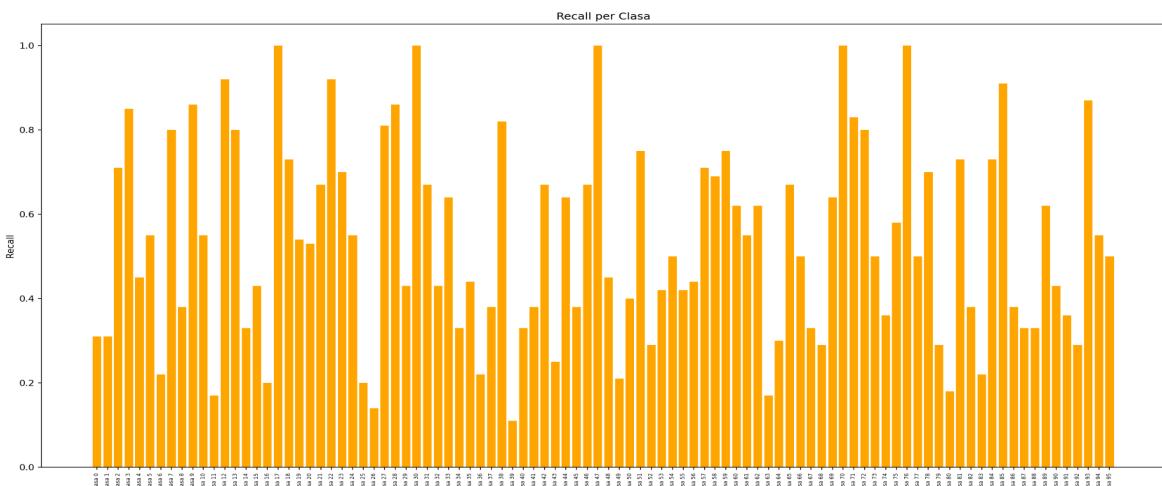
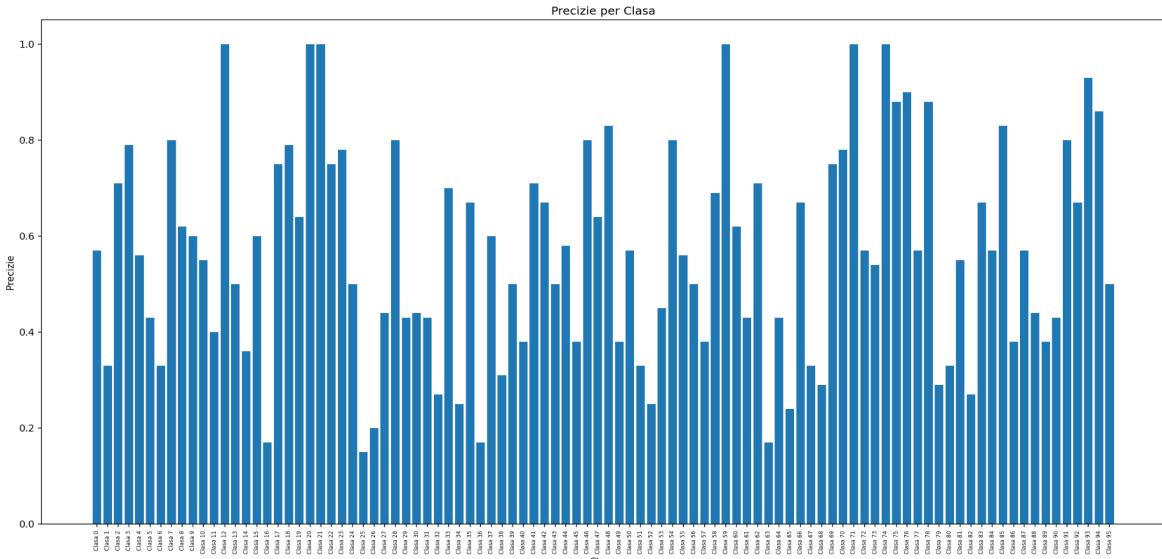
dar nu influențează acuratețea.

* Parametrul gamma setat la valori precum 0.1, 1, 10 duce la o acuratețe <0.1

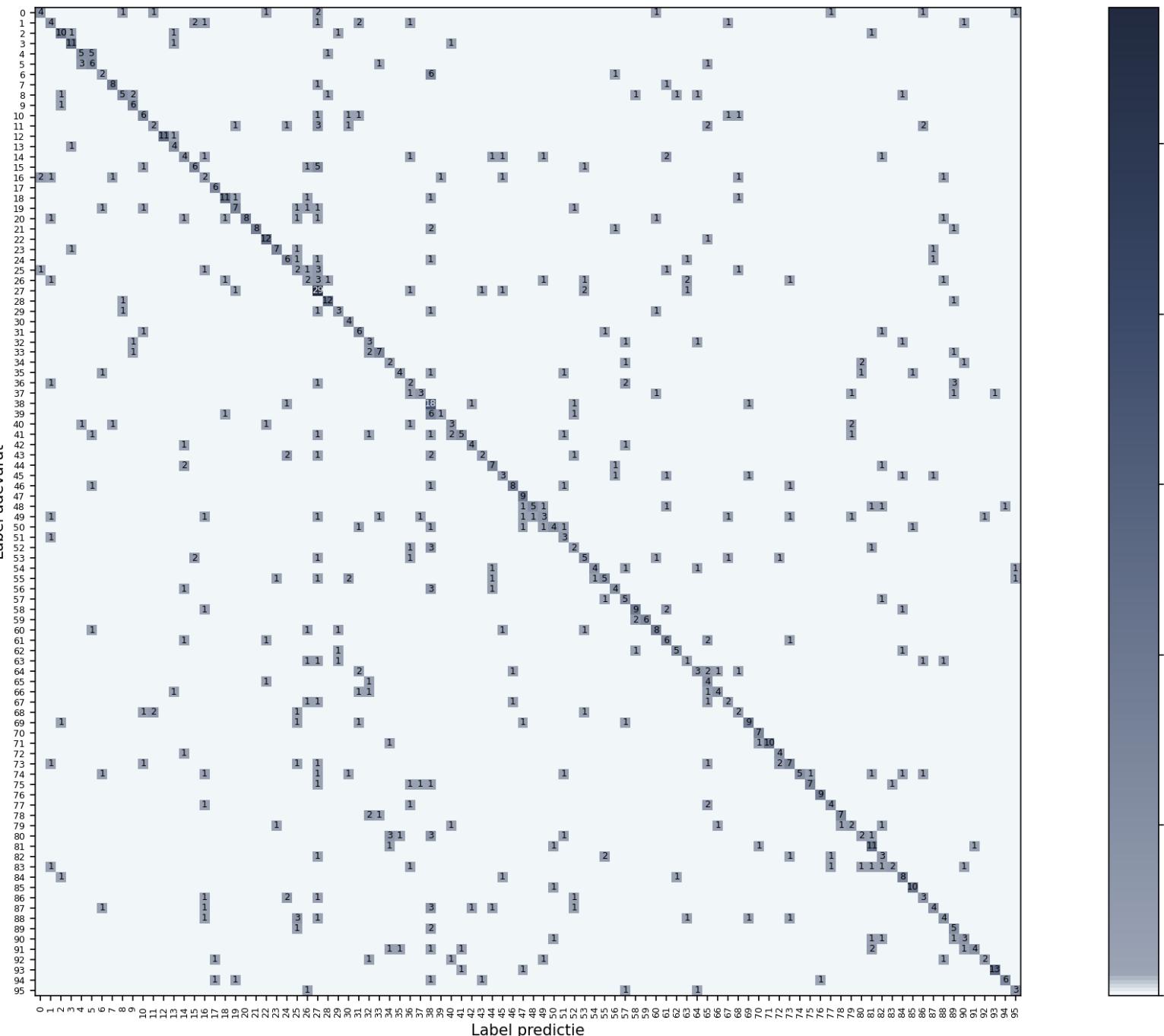
> Concluzie SVM

Cu implementarea svm: SVC(C=10, kernel='rbf', gamma='auto'), folosită pe feature set reprezentat de raw pixels representation + hog (orientations=9, pixels_per_cell=(8,8), cells_per_block=(2,2), channel_axis=-1) obținem următoarele:

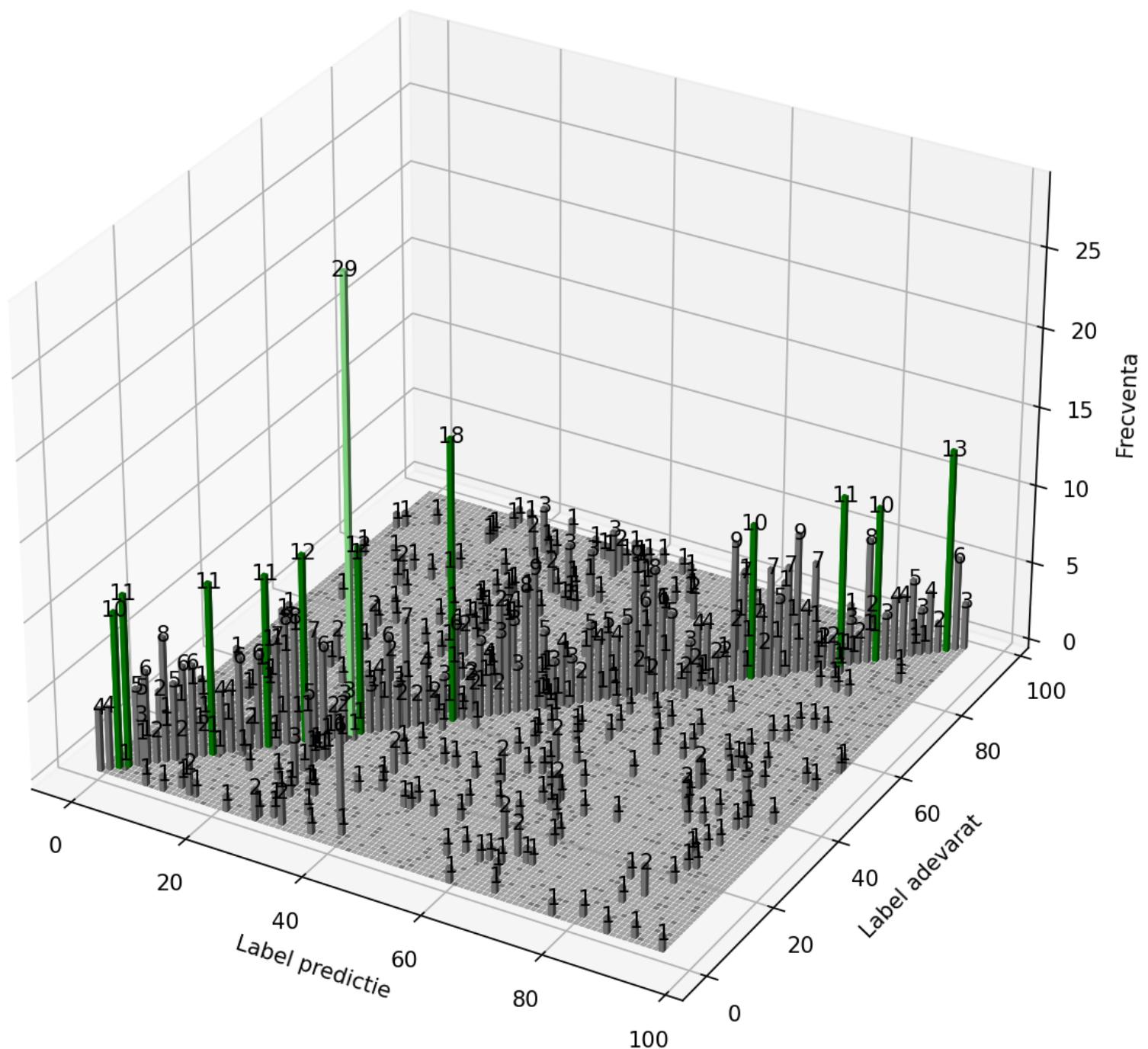
precision	recall	f1-score	support	Clasa 36	0.17	0.22	0.19	9	Clasa 75	0.88	0.58	0.70	12	
Clasa 0	0.57	0.31	0.40	13	Clasa 37	0.60	0.38	0.46	8	Clasa 76	0.90	1.00	0.95	9
Clasa 1	0.33	0.31	0.32	13	Clasa 38	0.31	0.82	0.45	22	Clasa 77	0.57	0.50	0.53	8
Clasa 2	0.71	0.71	0.71	14	Clasa 39	0.50	0.11	0.18	9	Clasa 78	0.88	0.70	0.78	10
Clasa 3	0.79	0.85	0.81	13	Clasa 40	0.38	0.33	0.35	9	Clasa 79	0.29	0.29	0.29	7
Clasa 4	0.56	0.45	0.50	11	Clasa 41	0.71	0.38	0.50	13	Clasa 80	0.33	0.18	0.24	11
Clasa 5	0.43	0.55	0.48	11	Clasa 42	0.67	0.67	0.67	6	Clasa 81	0.55	0.73	0.63	15
Clasa 6	0.33	0.22	0.27	9	Clasa 43	0.50	0.25	0.33	8	Clasa 82	0.27	0.38	0.32	8
Clasa 7	0.80	0.80	0.80	10	Clasa 44	0.58	0.64	0.61	11	Clasa 83	0.67	0.22	0.33	9
Clasa 8	0.62	0.38	0.48	13	Clasa 45	0.38	0.38	0.38	8	Clasa 84	0.57	0.73	0.64	11
Clasa 9	0.60	0.86	0.71	7	Clasa 46	0.80	0.67	0.73	12	Clasa 85	0.83	0.91	0.87	11
Clasa 10	0.55	0.55	0.55	11	Clasa 47	0.64	1.00	0.78	9	Clasa 86	0.38	0.38	0.38	8
Clasa 11	0.40	0.17	0.24	12	Clasa 48	0.83	0.45	0.59	11	Clasa 87	0.57	0.33	0.42	12
Clasa 12	1.00	0.92	0.96	12	Clasa 49	0.38	0.21	0.27	14	Clasa 88	0.44	0.33	0.38	12
Clasa 13	0.50	0.80	0.62	5	Clasa 50	0.57	0.40	0.47	10	Clasa 89	0.38	0.62	0.48	8
Clasa 14	0.36	0.33	0.35	12	Clasa 51	0.33	0.75	0.46	4	Clasa 90	0.43	0.43	0.43	7
Clasa 15	0.60	0.43	0.50	14	Clasa 52	0.25	0.29	0.27	7	Clasa 91	0.80	0.36	0.50	11
Clasa 16	0.17	0.20	0.18	10	Clasa 53	0.45	0.42	0.43	12	Clasa 92	0.67	0.29	0.40	7
Clasa 17	0.75	1.00	0.86	6	Clasa 54	0.80	0.50	0.62	8	Clasa 93	0.93	0.87	0.90	15
Clasa 18	0.79	0.73	0.76	15	Clasa 55	0.56	0.42	0.48	12	Clasa 94	0.86	0.55	0.67	11
Clasa 19	0.64	0.54	0.58	13	Clasa 56	0.50	0.44	0.47	9	Clasa 95	0.50	0.50	0.50	6
Clasa 20	1.00	0.53	0.70	15	Clasa 57	0.38	0.71	0.50	7	accuracy			0.54	1000
Clasa 21	1.00	0.67	0.80	12	Clasa 58	0.69	0.69	0.69	13	macro avg	0.57	0.53	0.53	1000
Clasa 22	0.75	0.92	0.83	13	Clasa 59	1.00	0.75	0.86	8	weighted avg	0.58	0.54	0.54	1000
Clasa 23	0.78	0.70	0.74	10	Clasa 60	0.62	0.62	0.62	13					
Clasa 24	0.50	0.55	0.52	11	Clasa 61	0.43	0.55	0.48	11					
Clasa 25	0.15	0.20	0.17	10	Clasa 62	0.71	0.62	0.67	8					
Clasa 26	0.20	0.14	0.17	14	Clasa 63	0.17	0.17	0.17	6					
Clasa 27	0.44	0.81	0.57	36	Clasa 64	0.43	0.30	0.35	10					
Clasa 28	0.80	0.86	0.83	14	Clasa 65	0.24	0.67	0.35	6					
Clasa 29	0.43	0.43	0.43	7	Clasa 66	0.67	0.50	0.57	8					
Clasa 30	0.44	1.00	0.62	4	Clasa 67	0.33	0.33	0.33	6					
Clasa 31	0.43	0.67	0.52	9	Clasa 68	0.29	0.29	0.29	7					
Clasa 32	0.27	0.43	0.33	7	Clasa 69	0.75	0.64	0.69	14					
Clasa 33	0.70	0.64	0.67	11	Clasa 70	0.78	1.00	0.88	7					
Clasa 34	0.25	0.33	0.29	6	Clasa 71	1.00	0.83	0.91	12					
Clasa 35	0.67	0.44	0.53	9	Clasa 72	0.57	0.80	0.67	5					
					Clasa 73	0.54	0.50	0.52	14					
					Clasa 74	1.00	0.36	0.53	14					



Matrice de confuzie 2D



Matrice de confuzie 3D



➤ Convolutional Neural Network (CNN)

Pentru CNN mă voi folosi de raw pixels representation. Pornesc cu un model simplu, două straturi convoluționale (kernel_size=3, padding=1), fiecare cu câte un strat maxpool (kernel și stride = 2), iar la final un strat dens. Numărul de channel-uri: 3 -> 32 -> 64 -> 128 -> 96. Pentru loss function folosesc **CrossEntropyLoss()** din pytorch iar ca optimizer folosesc **Adam** cu un learning rate de 0.001. Momentan, fără data augmentation, cu shuffle pe datele de antrenare. Pentru straturile convoluționale am folosit activare **ReLU()**. Batch size-ul este 64.

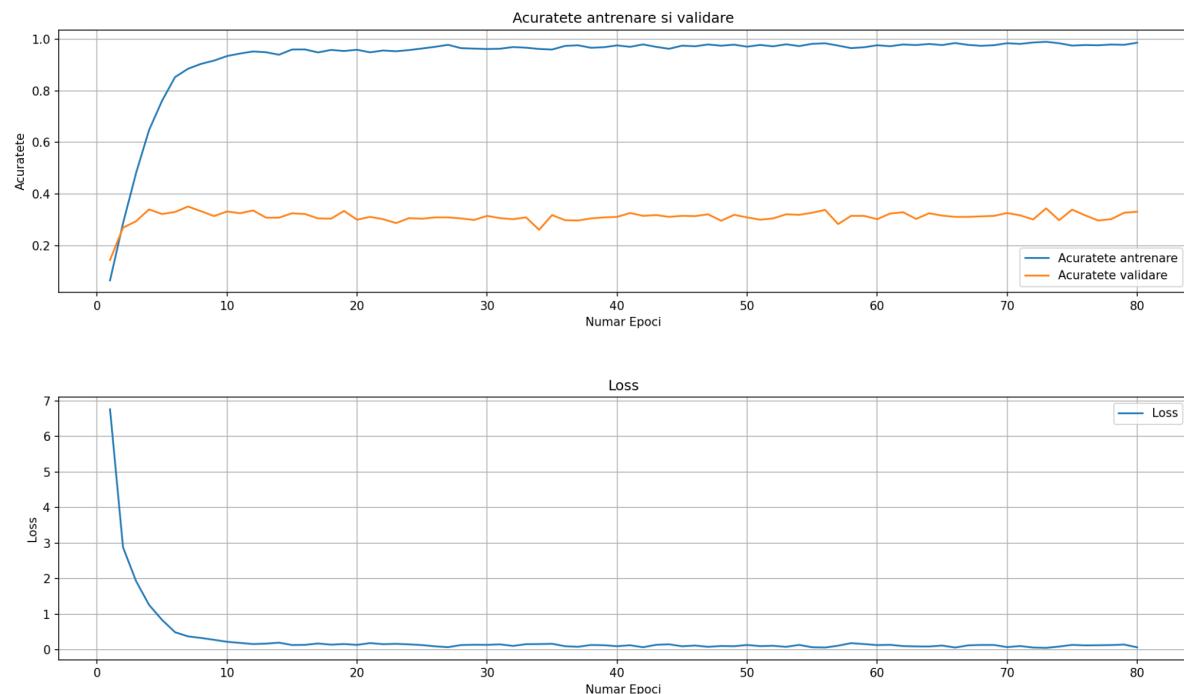
❓ De ce CrossEntropyLoss()

➡ CrossEntropyLoss este o combinație de **Softmax Activation** și **Negative Log-Likelihood Loss**. Softmax reprezintă funcția aplicată pe output-ul ultimului strat pentru a converti valorile de output în probabilități după care Negative Log-Likelihood Loss calculează loss-ul ca fiind $-\log(\text{probabilitate label adevărat})$, astfel cu cât probabilitatea este mai mică, cu atât loss-ul este mai mare. În total, loss-ul este calculat ca fiind media dintre acești logaritmi pe un anumit batch.

❓ De ce Adam

➡ Adam (Adaptive Moment Estimation) este un optimizer popular, și l-am ales doar pentru o rulare inițială, urmează a fi comparat cu alte optimizere.

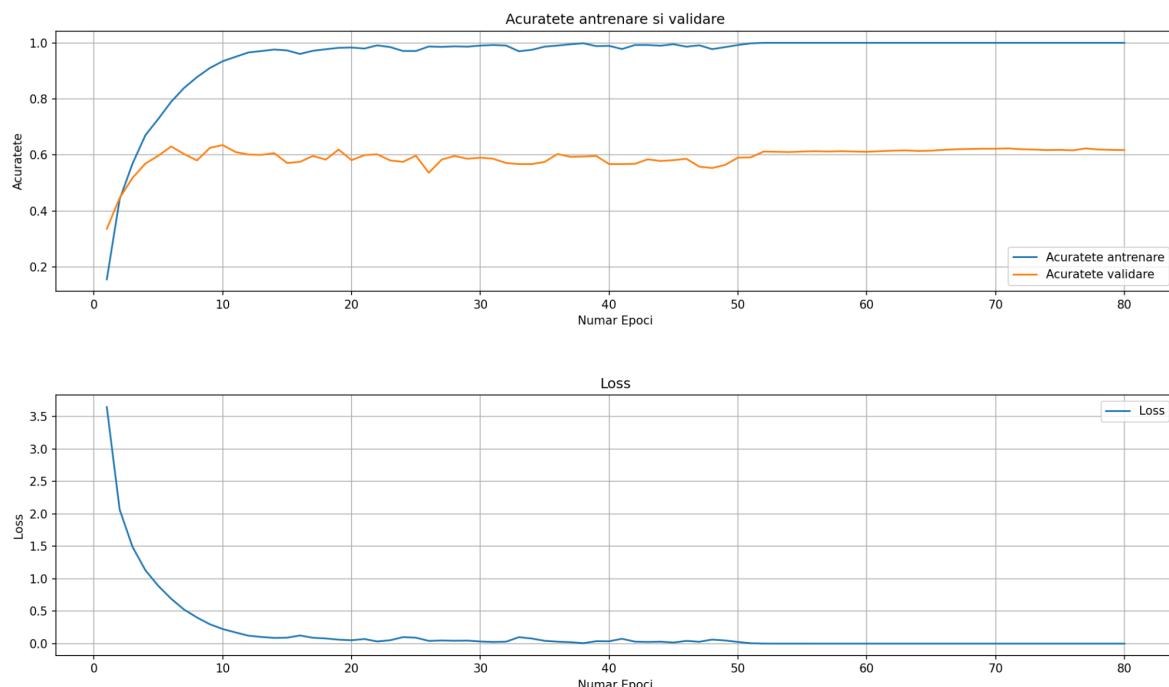
La o prima rulare pe un număr arbitrar de epoci (80) avem următoarele rezultate:



Acuratețea foarte mare pe antrenare dar mică pe validare sugerează faptul că modelul învață bine dar generalizează prost (overfit).

➤ Introducем нормализацию

Se recomandă împărțirea valorilor pixelilor la 255 pentru a reduce complexitatea de calcul și a face modelul mai rapid ([Inspirat de aici](#)).

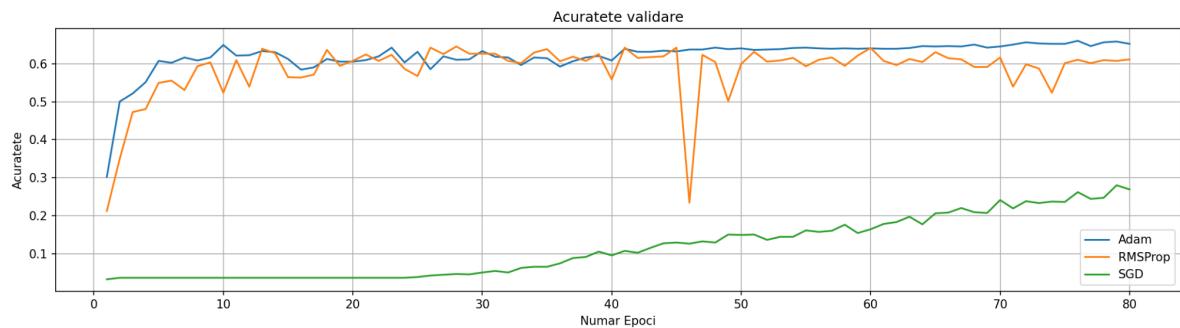
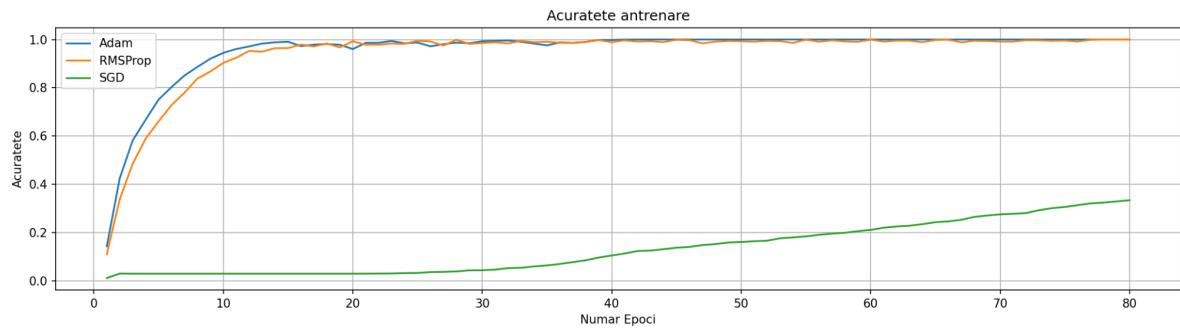


Deși overfit-ul se menține, observăm că totuși generalizează mai bine, acuratețea ajungând la 0.623 în loc de 0.332.

➤ Сравнение Adam, SGD, RMSprop

Din graficul de mai jos se observă că atât Adam cât și RMSprop au o învățare foarte rapidă, și o generalizare la fel de rapidă. Totuși, Adam pare să fie puțin mai stabil decât RMSprop.

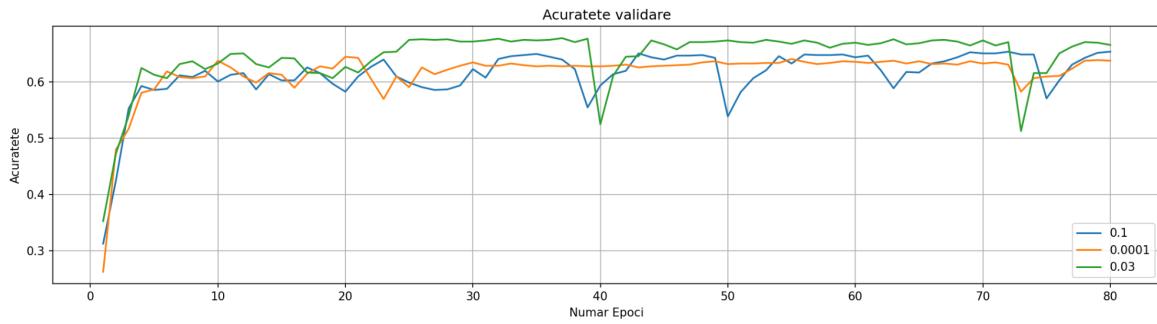
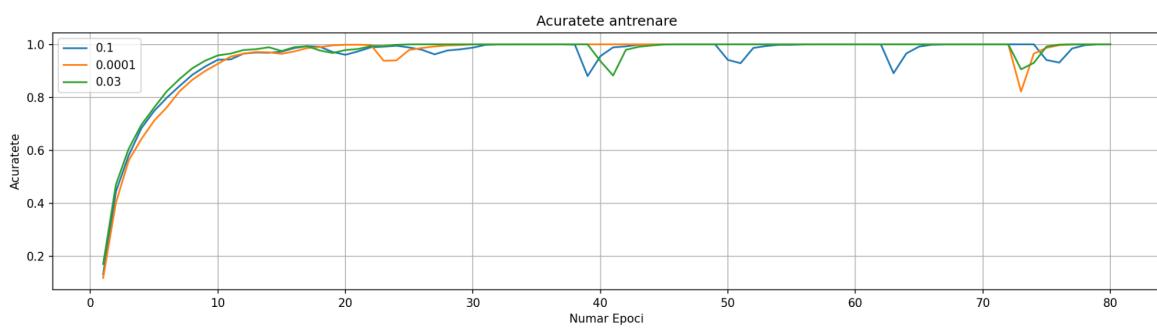
Chiar dacă SGD are o creștere mai stabilă și echilibrată între antrenare și validare, aceasta se realizează foarte lent, motiv pentru care prefer să rămân cu Adam.



➤ Weight decay

Pentru a descuraja overfitting-ul voi încerca să aplic weight decay care nu permite modelului să se bazeze pe anumite weights care tind să ajungă foarte mari, deoarece acestea se degradează în timp. Ca să pot aplica weight decay pe Adam, voi folosi AdamW. Astfel, se realizează o regularizare L2.

Voi folosi `weight_decay=0.03`

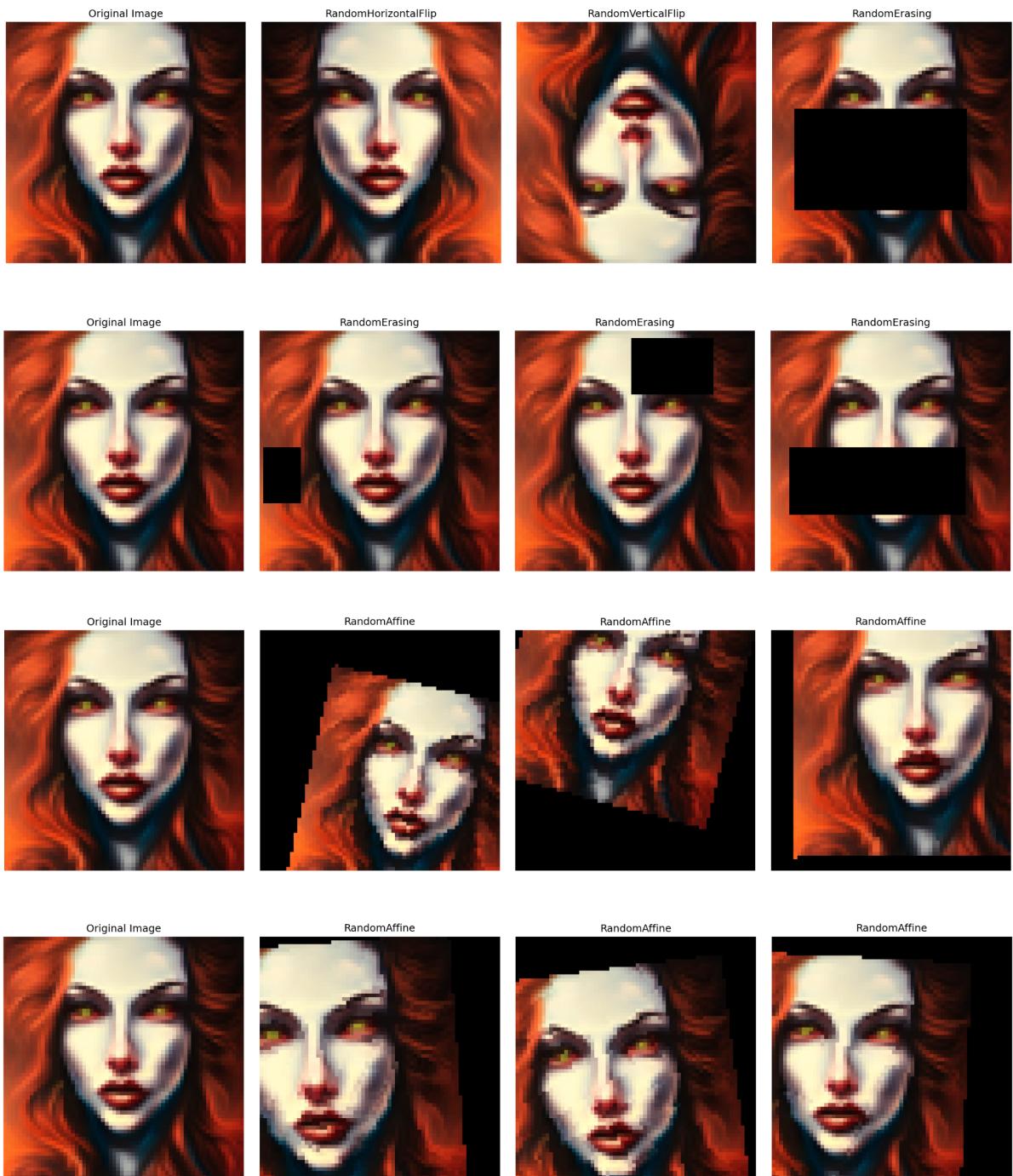


➤ Data Augmentation

Tot în vederea reducerii overfit-ului și pentru a îmbunătății puterea de generalizare a modelului, voi introduce transformări din pytorch.

Vizualizare a diverse transformări de unde m-am inspirat: [Aici](#)

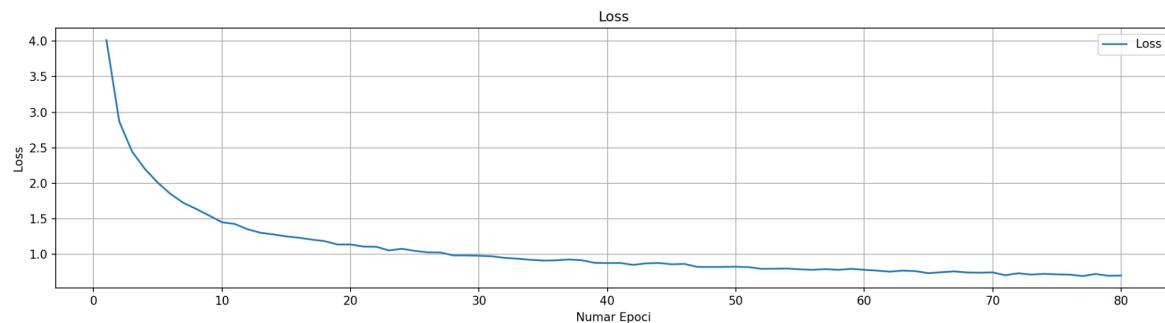
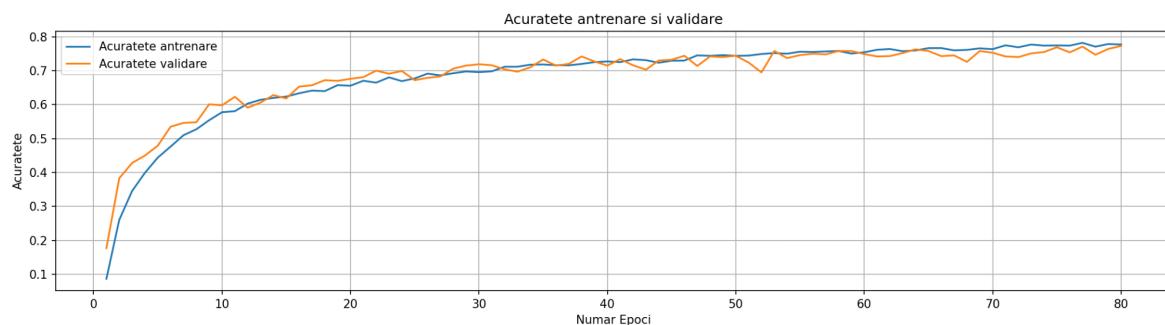
Imaginea 00000.png din setul de antrenare:



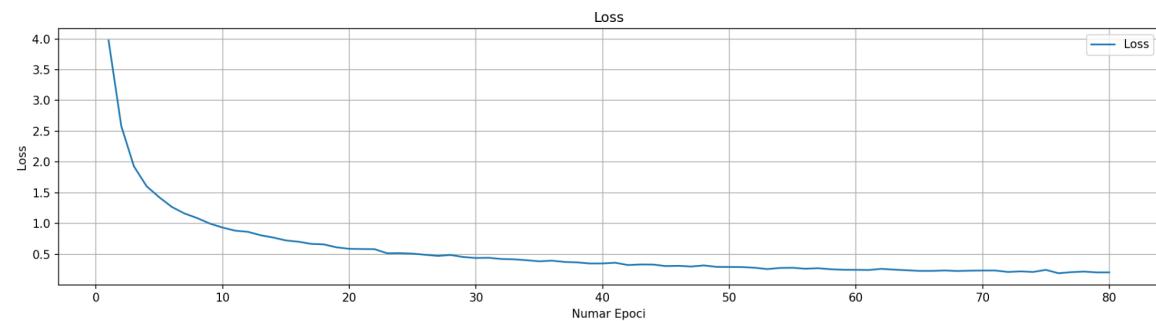
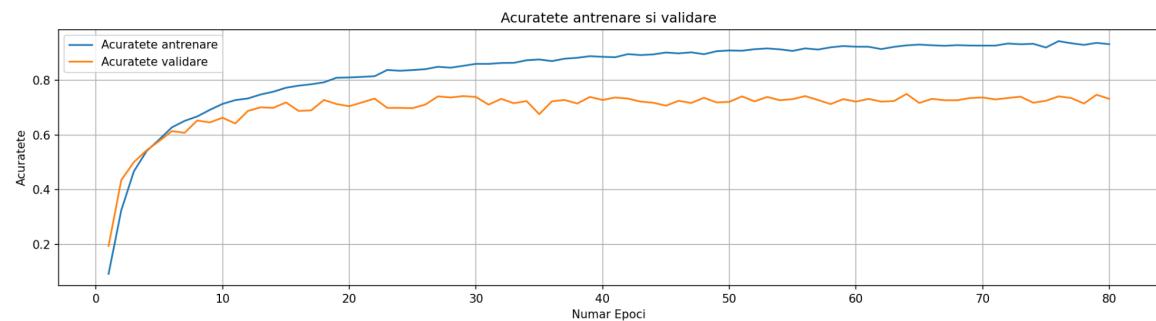
Aplicăm astfel următoarele transformări:

```
transform=transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomAffine(degrees=15, scale=(0.75, 1.25), shear=0.25),
    transforms.RandomErasing()
])
```

În urma adăugării lor, timpul de rulare crește foarte mult, dar la fel și puterea de generalizare, care acum este proporțională cu cea de învățare. Obține o acuratețe de 0.733 pe validare!

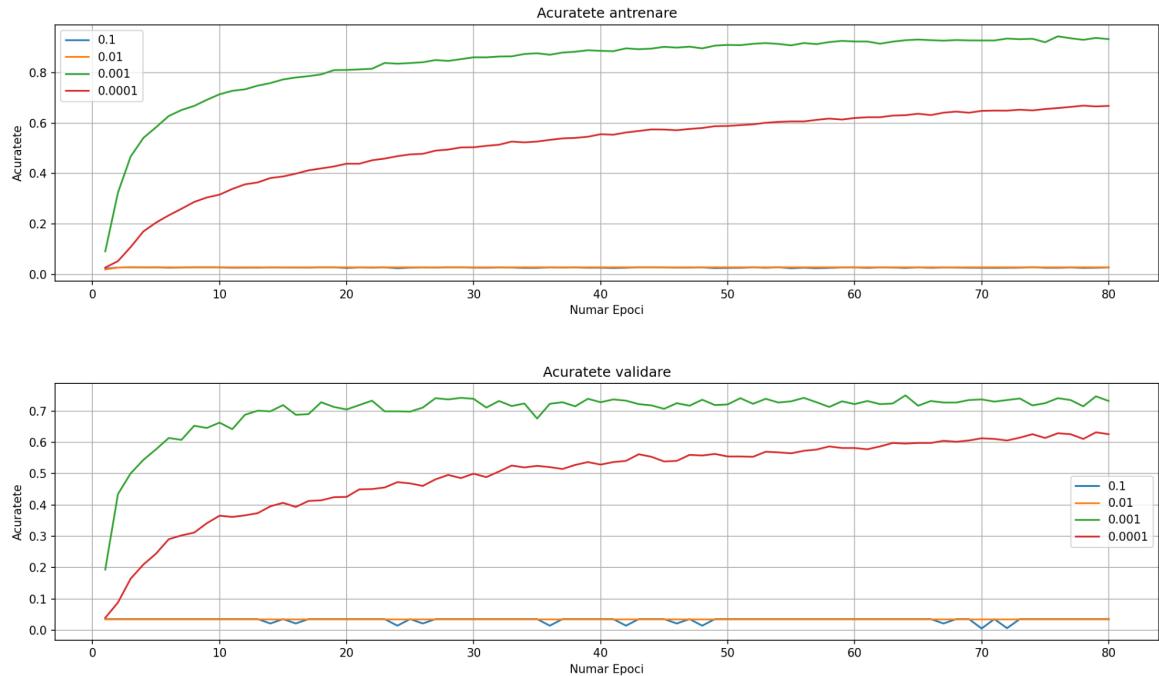


Pentru a antrena mai rapid modelul voi îndepărta momentan RandomAffine, care pare a fi cauza procesării lente. În lipsa lui se obține aceeași acuratețe doar că modelul învăță mai rapid decât poate generaliza.

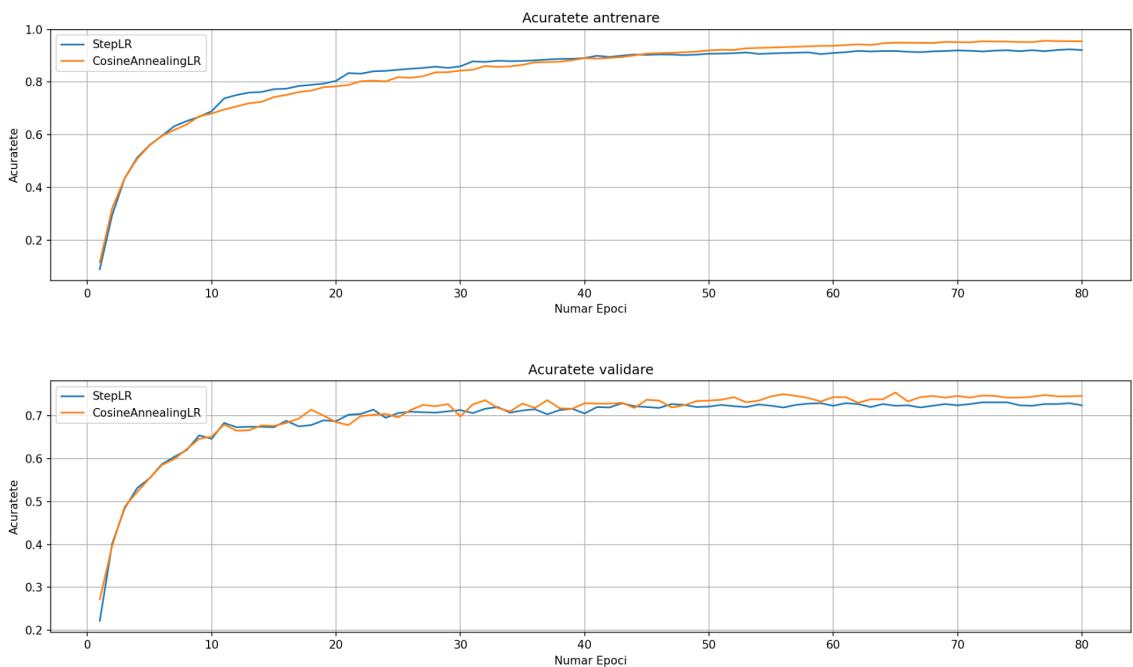


➤ Learning Rate & Learning Rate Scheduler

Experimentez folosind un learning rate diferit, ajung să-l păstreze pe cel ales deja (0.001):

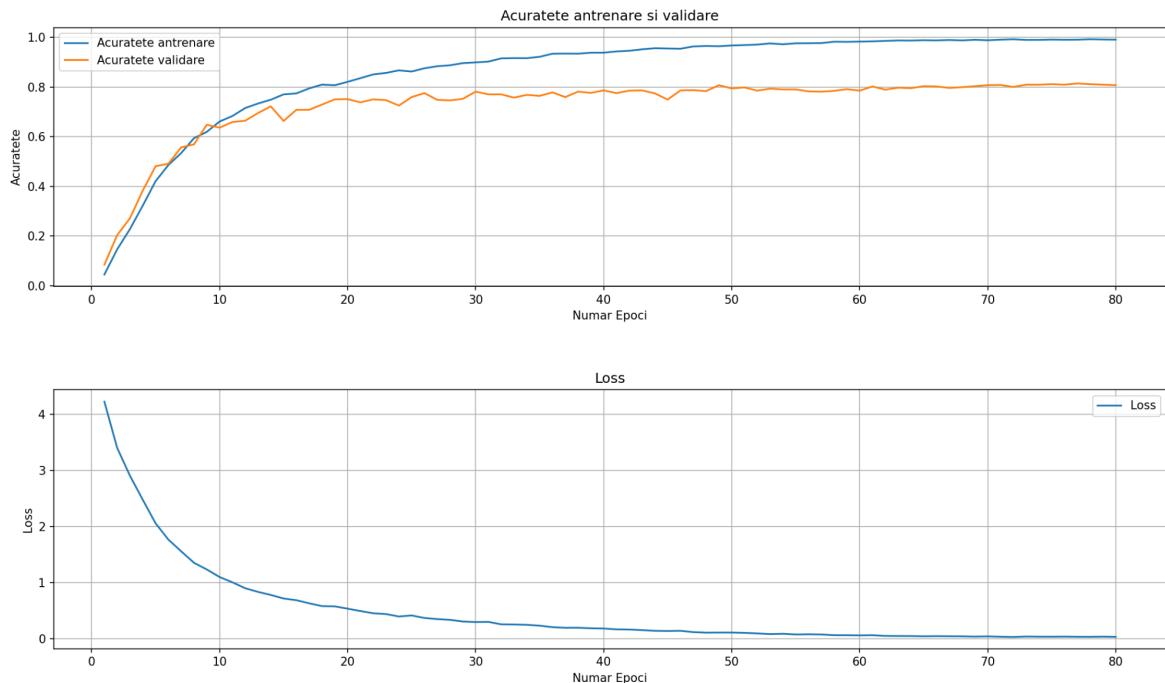


Folosind un learning rate scheduler putem să îmbunătățim generalizarea modelului. Aleg între StepLR și CosineAnnealingLR, schedulere care nu necesită parametru pentru step(). Rămân cu CosineAnnealingLR.



➤ Model Upgrade

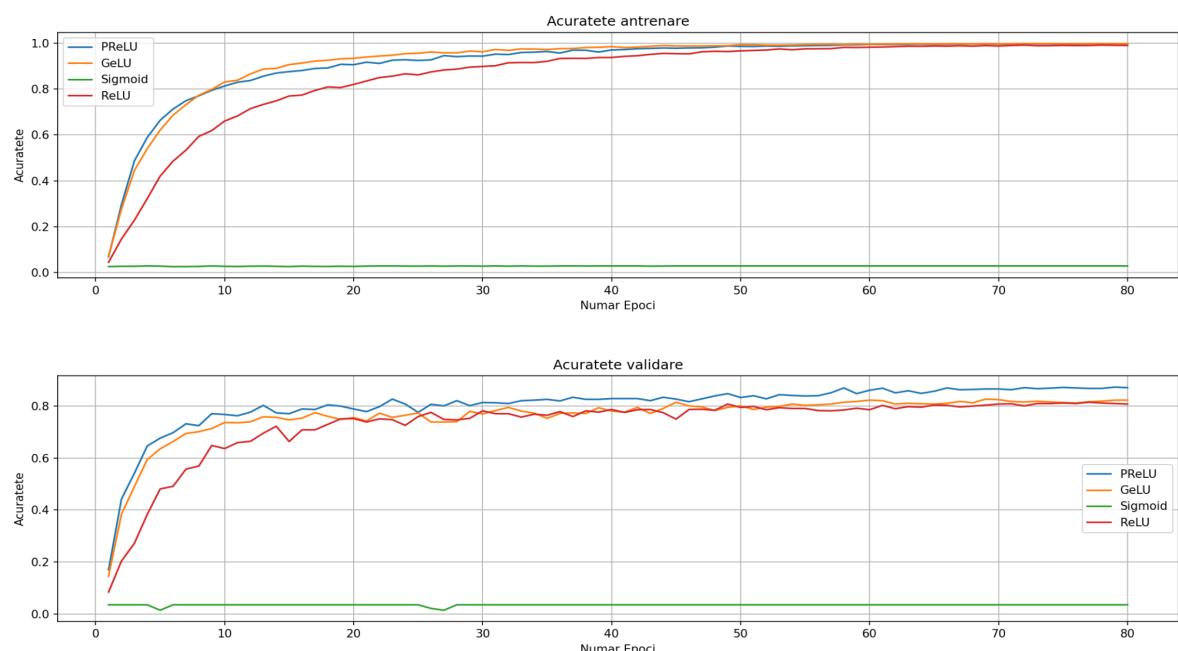
În acest moment cred că modelul începe să își atingă capacitatea maximă pe care o are cu straturile sale, motiv pentru care doresc să îl transform în: 6 straturi conveționale, cu maxpool din două în două straturi, pornind de la al doilea strat. Păstrează configurația tuturor conveționalilor la `kernel_size=3`, `padding=1`, `stride=1`, și a maxpool-urilor la `stride=kernel_size=2`. Acum modelul are următoarele channel-uri: 3 -> 32 -> 64 -> 64 -> 64 -> 128 -> 256 -> 96



Cu acest upgrade, acuratețea ajunge la 0.8! Dar nu este suficient...

➤ Activation Function

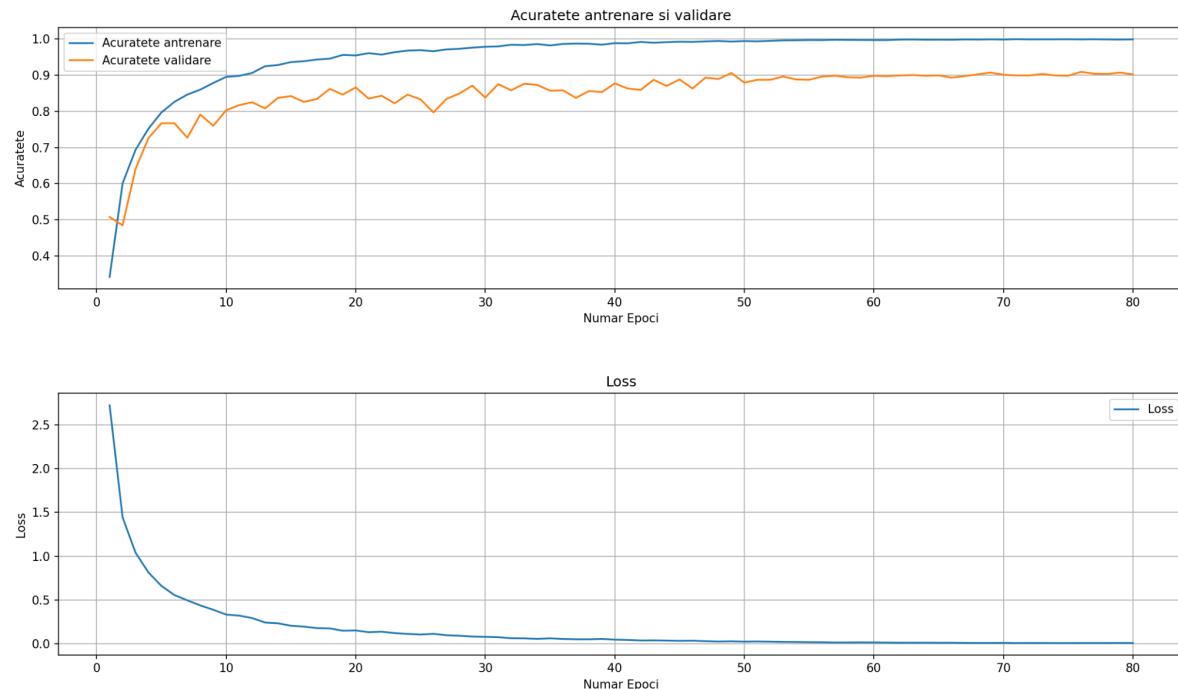
Până acum am folosit ReLU, dar ar trebui să verific cum se comportă și cu alte funcții de activare.



➤ Batch Normalization

Folosesc batch normalization pentru a stabiliza modelul și a-l face mai puțin sensibil la alegerile de learning rate. Ajută la gradient flow și reprezintă o formă de regularizare a imaginilor, adăugând noise, deci ajută și în cazul overfitting-ului.

În urma aplicării normalizării pe toate straturile de conoluție cât și pe cel dens, se obține o acuratețe de validare de 0.9!



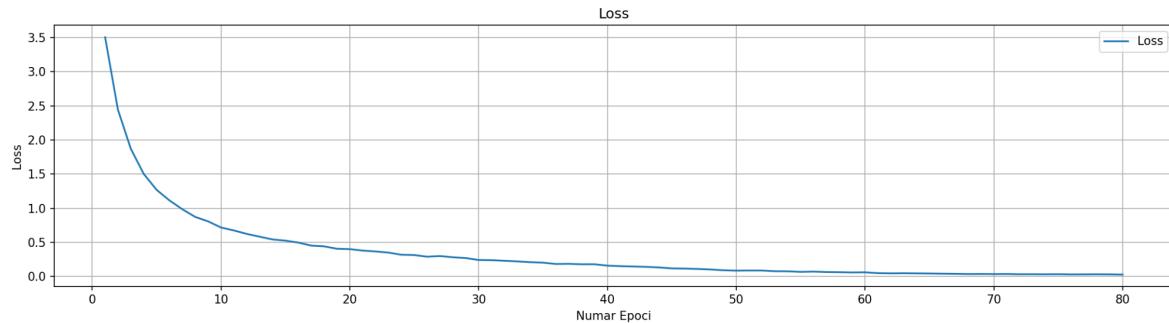
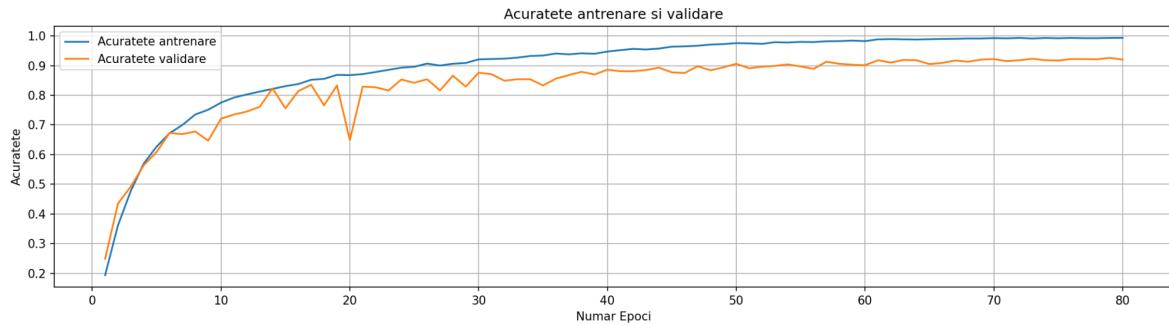
Adăugând încă un strat dens și modificând numărul de channel-uri este acum: 32 -> 64 -> 64 -> 64 -> 64 -> 128 -> 256 -> 64 -> 96, care pare să îmbunătățească foarte puțin (de la 0.9 la 0.91).

➤ Model upgrade 2

Mai adaug 3 straturi conoluționale, respectând același pattern (kernel_size=3, stride=1, padding=1), cu un strat maxpool după ultimul, și cu batch normalization după fiecare strat conoluțional. Mențin tendința ascendentă a numărului de channel-uri pe straturile conoluționale, și obțin: 32 -> 64 -> 64 -> 64 -> 64 -> 128 -> 128 -> 256 -> 512 -> 96

Folosesc ascending în loc de bottleneck deoarece nu am un număr foarte mare de straturi, care să necesite reducerea costului de computație.

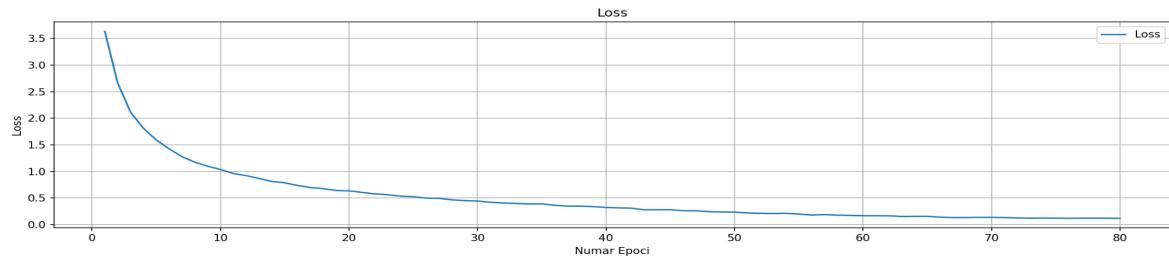
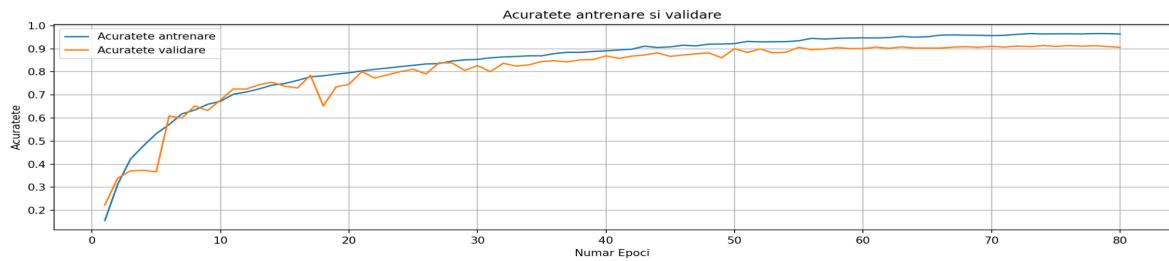
În urma acestor modificări, acuratețea crește la 0.92!



De asemenea, o să încerc să modific kernel size-ul ultimului strat conoluțional de la 3 la 5, întrucât aş vrea să capteze informații cu o imagine de ansamblu puțin mai mare, asupra feature-urilor extrase până în acel punct.

➤ Revine RandomAffine

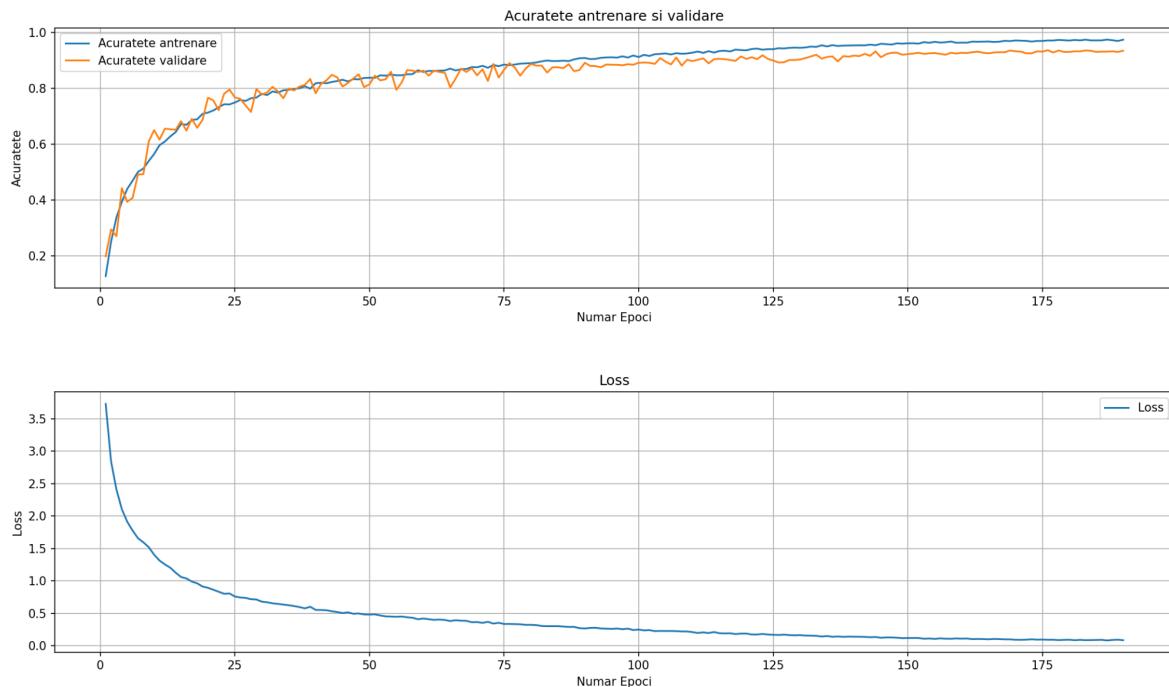
Întrucât consider că modelul a ajuns într-un punct destul de stabil, adaug RandomAffine din nou pentru a vedea dacă puterea de generalizare o să se apropie mai mult de cea de antrenare.



Capacitatea de învățare este mai aproape de cea de generalizare, dar capacitatea de generalizare nu a crescut.

➤ Mai multe epoci

Deși la pasul anterior nu am avut îmbunătățiri, pare ca ar putea să crească. Cresc numărul de epoci la 190.

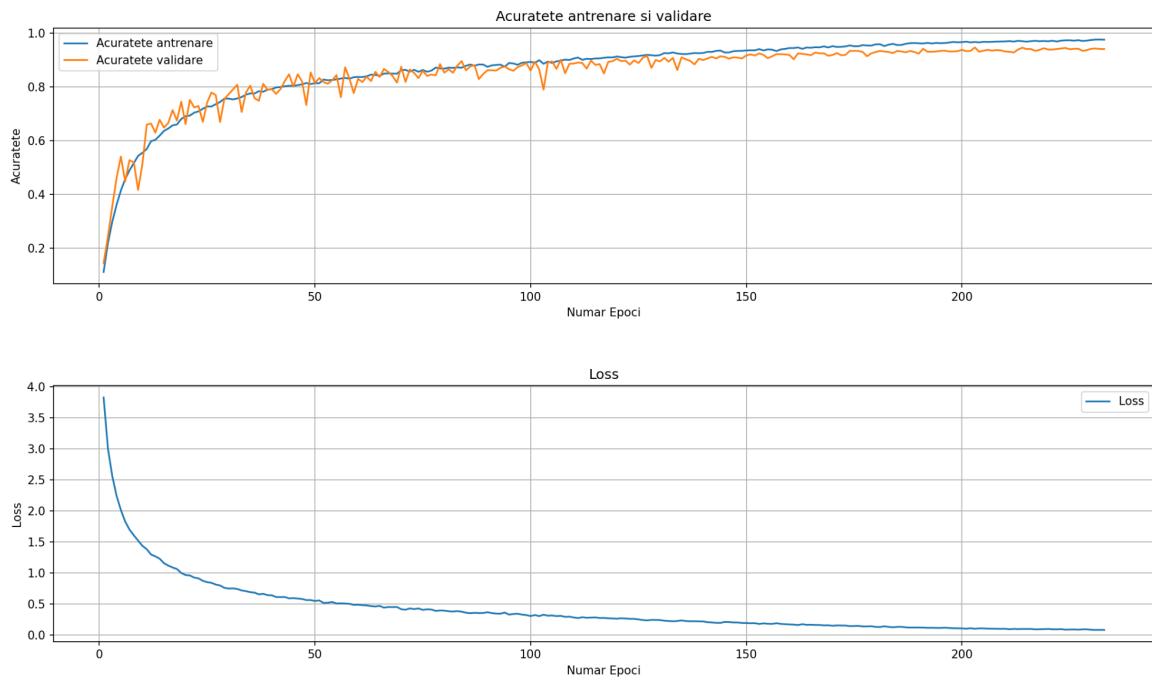


Acum se obține acuratețea pe validare de 0.93!

➤ Early Stopping

Pentru a preveni overfitting-ul și pentru a nu îl lăsa să continue inutil fără îmbunătățiri introduc mecanismul de early stopping cu patience=30, în funcție de acuratețea de validare.

Încerc să mai măresc numărul de epoci iar la 250, dar de data asta early stopping-ul îl oprește la epoca 233. Modelul obține acum acuratețea 0.946 pe validare, dar pe datele de test (Kaggle) obține doar 0.918, ceea ce sugerează că încă există overfitting.

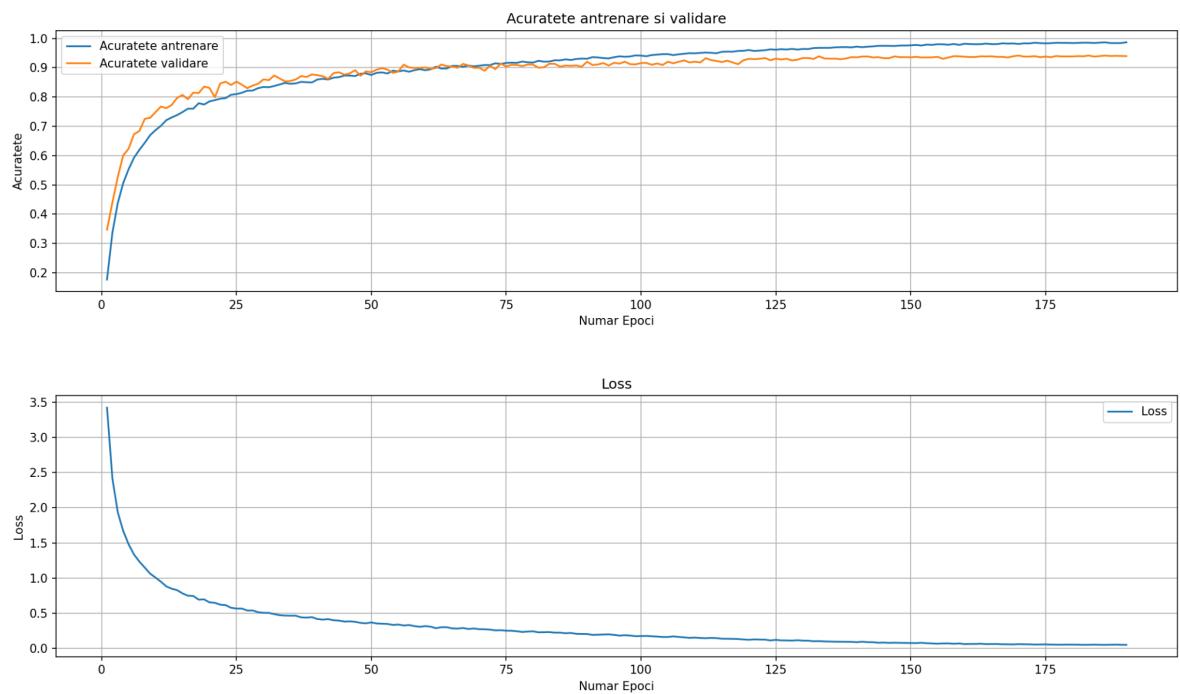


➤ Oportunitate - Folosirea datelor de testare de la competiția anulată

Întrucât setul de date este relativ redus, consider că modelul atinge o capacitate de generalizare cauzată de insuficiență datelor. Astfel, profit de faptul că ultimele două cifre din denumirea imaginilor reprezintă clasa imaginii. Acest lucru se întâmplă în asociere directă pentru clasele 0 - 52, după care de la clasa 53 începe o ușoară decalare a clasei corespondente (dacă se termină în 53 aparține clasei 54, în 54 clasei 56, etc). Există în continuare posibilitatea să existe și clasificări greșite, prezente accidental sau intenționat, dar per total imaginile acestea de test oferă informații corecte.

Deși obține acuratețea 0.9415 pe teste de validare, pe Kaggle obține 0.943! Clar o îmbunătățire și o reducere a overfitting-ului.

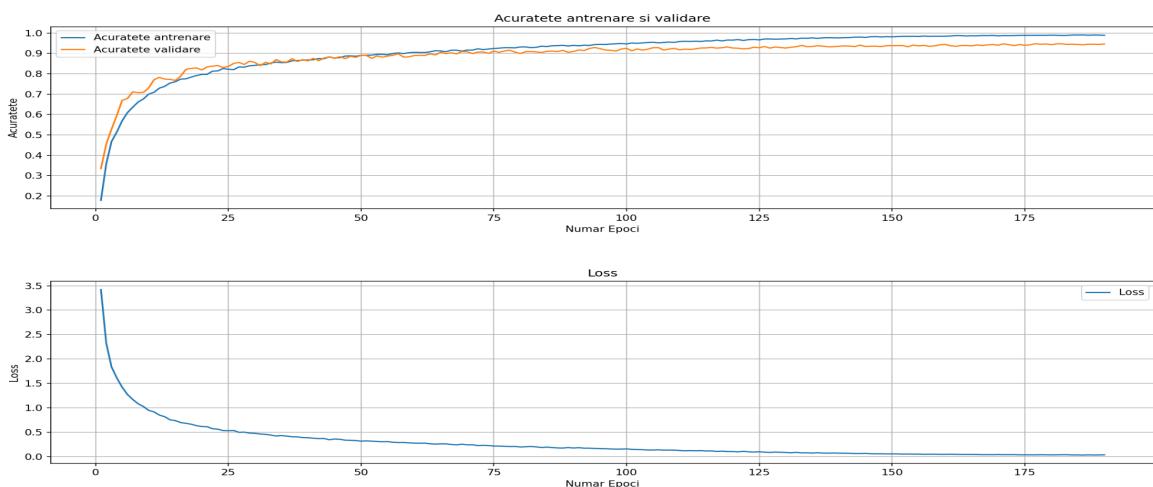
De asemenea, se observă pe grafic că acuratețea pe validare are spike-uri mult mai reduse.



➤ Skip Layer/Connection

Folosirea de skip layers facilitează flow-ul gradientului deoarece oferă mai multe locuri prin care gradientul poate avansa. Încurajează reutilizarea feature-urilor care devin comprimate pe parcursul a mai multor straturi. De asemenea, ajută la ameliorarea efectului de vanishing gradient. Efectul de vanishing gradient se referă la faptul că straturile de la început primesc puține actualizări atunci când are loc backpropagation din straturile de la final.

Adaug un skip connection de la primul strat convecțional la cel de-al optulea. Folosesc downsampling printr-un AdaptiveMaxPool și o convecție cu `kernel_size=3, stride=1, padding=0` pentru a potrivi dimensiunile spațiale și numărul de channel-uri.



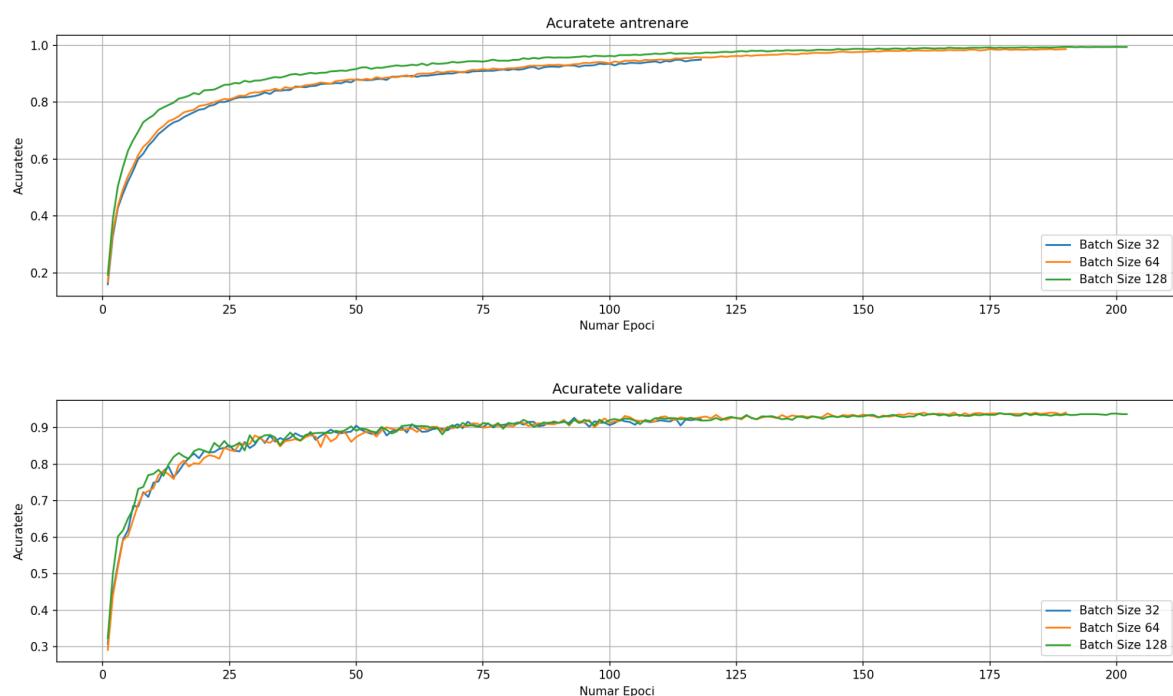
Deși obține 0.945 acuratețe de validare, pe Kaggle obține 0.9533 (pe 30% din teste, și 0.93866 pe 70% din teste).

➤ Alte încercări pe CNN

Din stadiul obținut la pasul de skip layer, am mai încercat diverse modificări, care însă nu au îmbunătățit notabil, unele chiar au înrăutățit.

➤ Modificare Batch Size

Nu se observă diferențe substanțiale între 32, 64, 128:

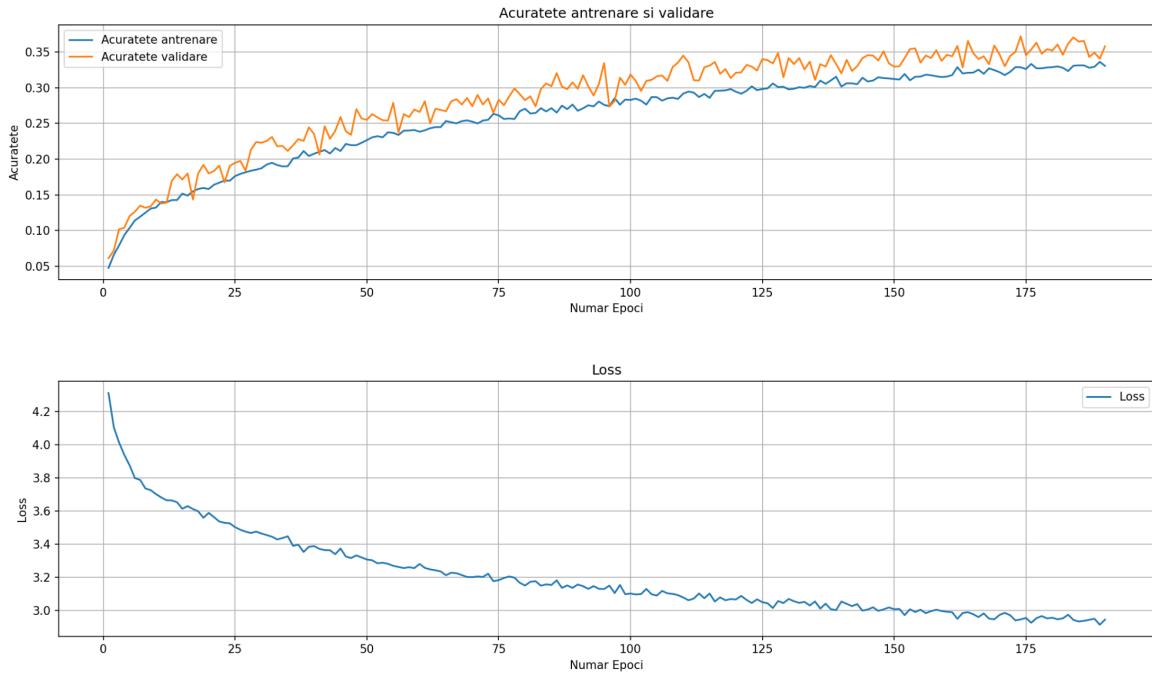


➤ Dropout

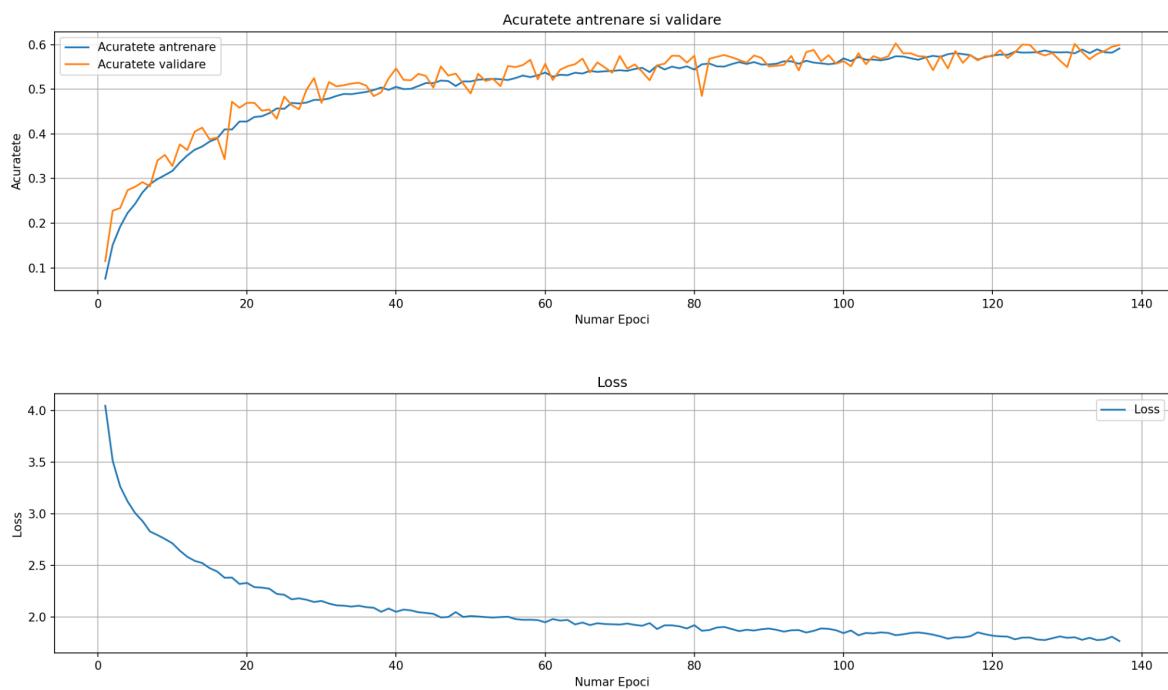
Dropout-ul este o tehnică de regularizare comună folosită în preventia overfitting-ului, dezactivând temporar neuroni în forward și back propagation, împiedicând astfel modelul să depindă prea mult de anumiți neuroni, și făcându-l să caute rute alternative. Deși sună bine, el nu performează bine pe acest model, acuratețea ajungând sub 0.70.

În cele două grafice de mai jos, s-a testat cu dropout cu probabilitate mică la fiecare strat (0.01, 0.03, 0.1, 0.1, 0.2, 0.2, 0.3, 0.3, 0.4, 0.4, 0.4) unde a obținut acuratețea de aproximativ 0.35, apoi cu dropout doar pe ultimele două straturi dense, unde a obținut acuratețea de aproximativ 0.6.

Dropout. Fig 1.

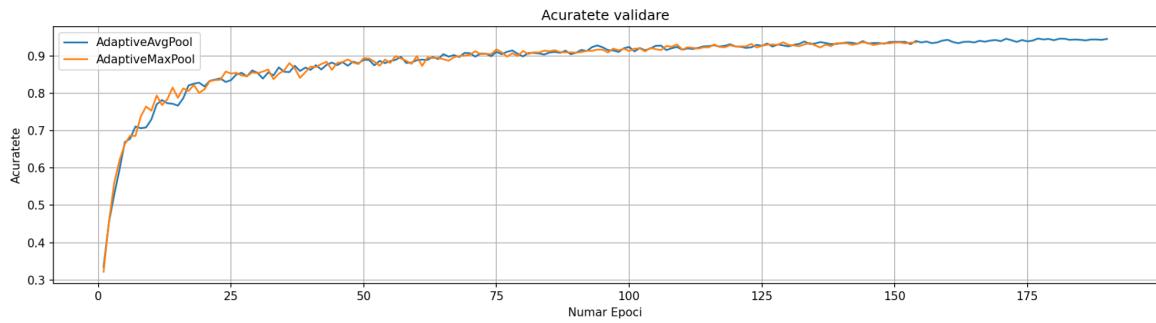
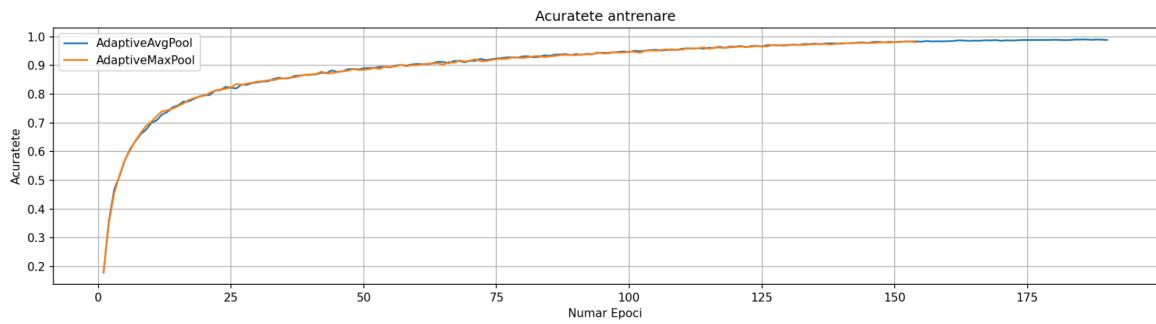


Dropout. Fig 2.

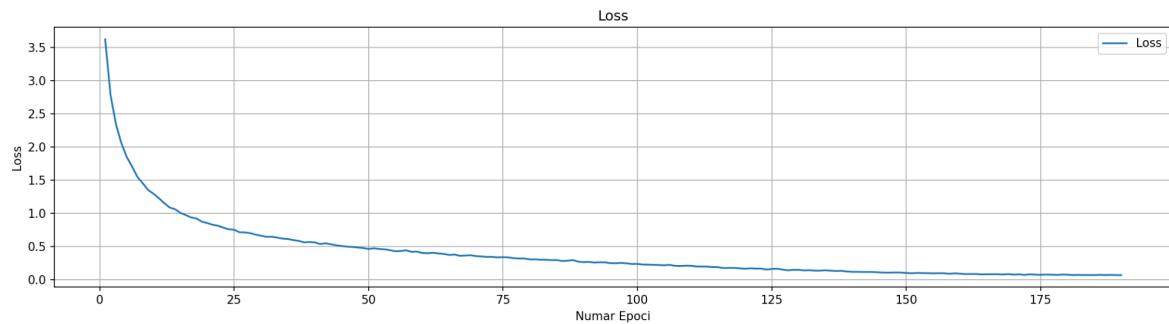
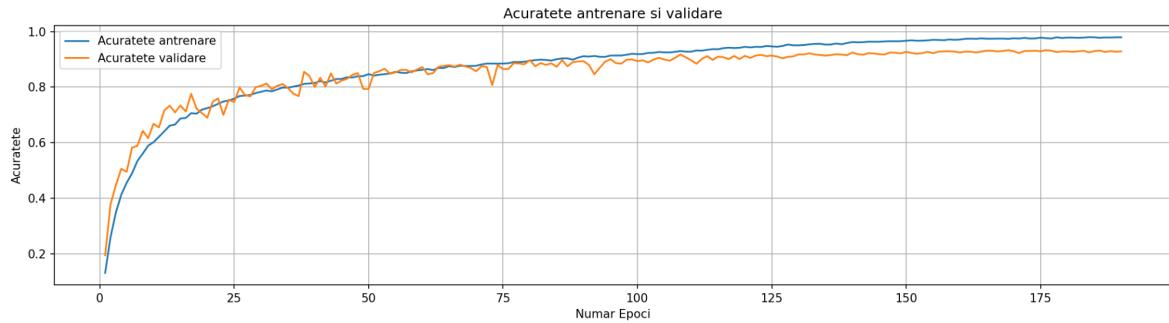


➤ AdaptiveMaxPool vs AdaptiveAvgPool

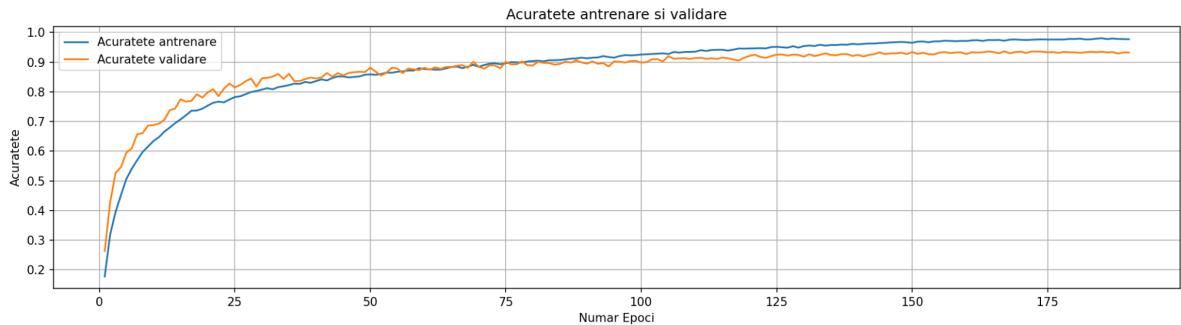
În skip layer-ul folosit, folosesc un adaptive pool pentru a echilibra dimensiunile spațiale ale celor două straturi care se adună. AdaptiveMaxPool cauzează early stopping-uri, motiv pentru care păstrează AdaptiveAvgPool.



Cu ocazia comparării dintre Avg și Max, am încercat și înlocuirea tuturor straturilor maxpool cu avgpool, dar am obținut o acuratețe mai mică, de 0.92.

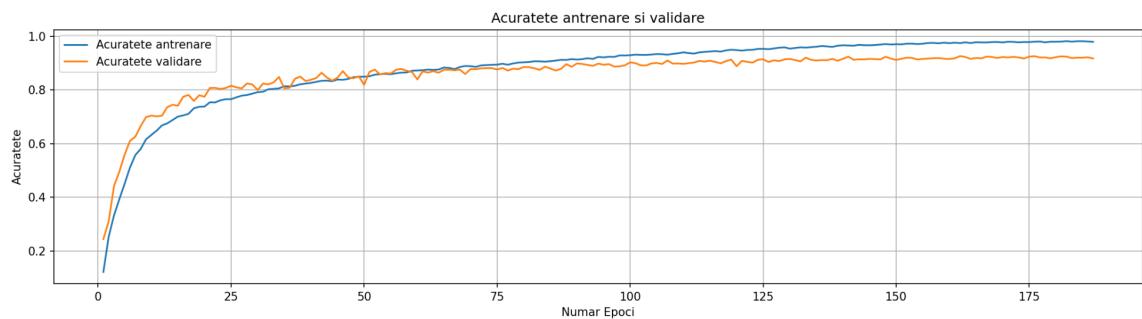


De asemenea, avgpool doar pe ultimele două straturi și în rest maxpool pare să coboare acuratețea de validare la 0.93 de asemenea.



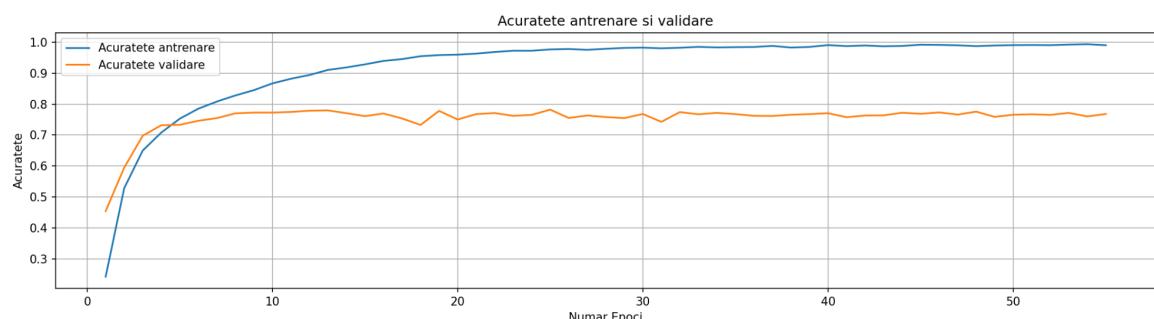
➤ RandomInvert

Transformarea RandomInvert coboară acuratețea la 0.92



➤ HOG Features

Întrucât la SVM s-au dovedit a fi utile, îmi încerc norocul și aici, în speranță că pot să mai diversific setul de antrenare, care este mediu ca mărime. Prin concatenare, imediat după flatten, măresc numărul de features, astfel încât straturile dense să poată folosi atât detaliile procesate din pixeli cât și hog. Această modificare nu pare să ajute, provocând early stopping destul de devreme.



➤ Experimente pe transformări

Diverse tentative de îmbunătățire, care nu au dat rezultate mai bune.

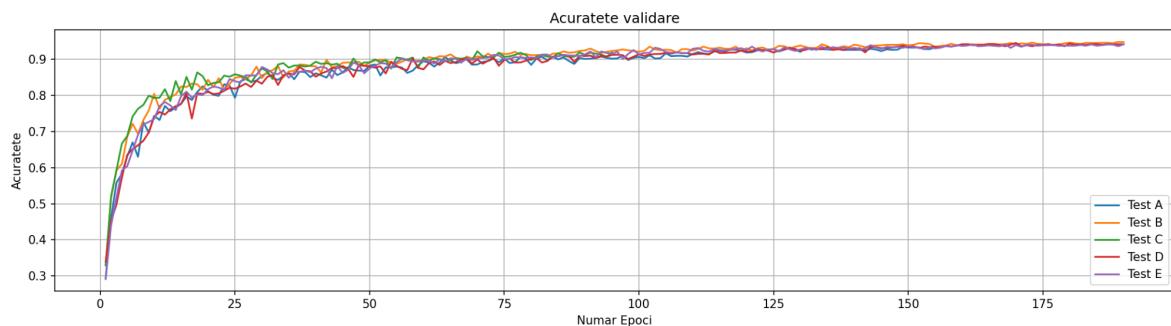
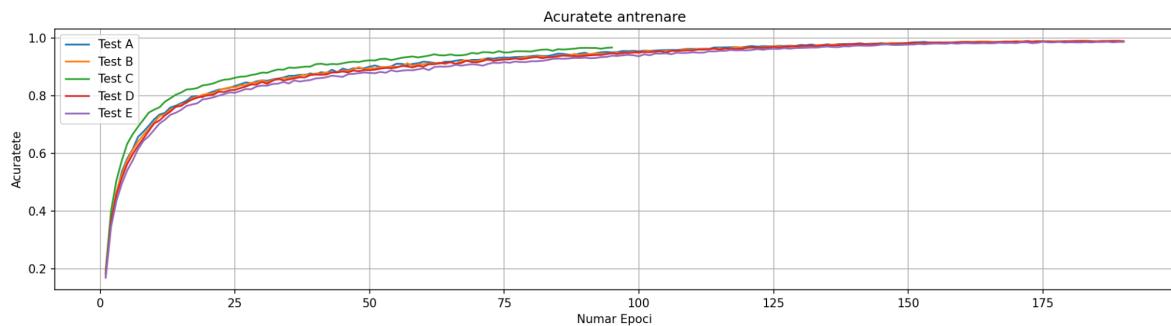
(0.933) **Test A:** Probabilitate de 0.75 pe RandomHorizontalFlip și RandomVerticalFlip

(0.944) **Test B:** Probabilitate 0.3 pe RandomVerticalFlip, 0.5 pe RandomHorizontalFlip

(0.919) **Test C:** Doar RandomHorizontalFlip de 0.5, fără RandomVerticalFlip

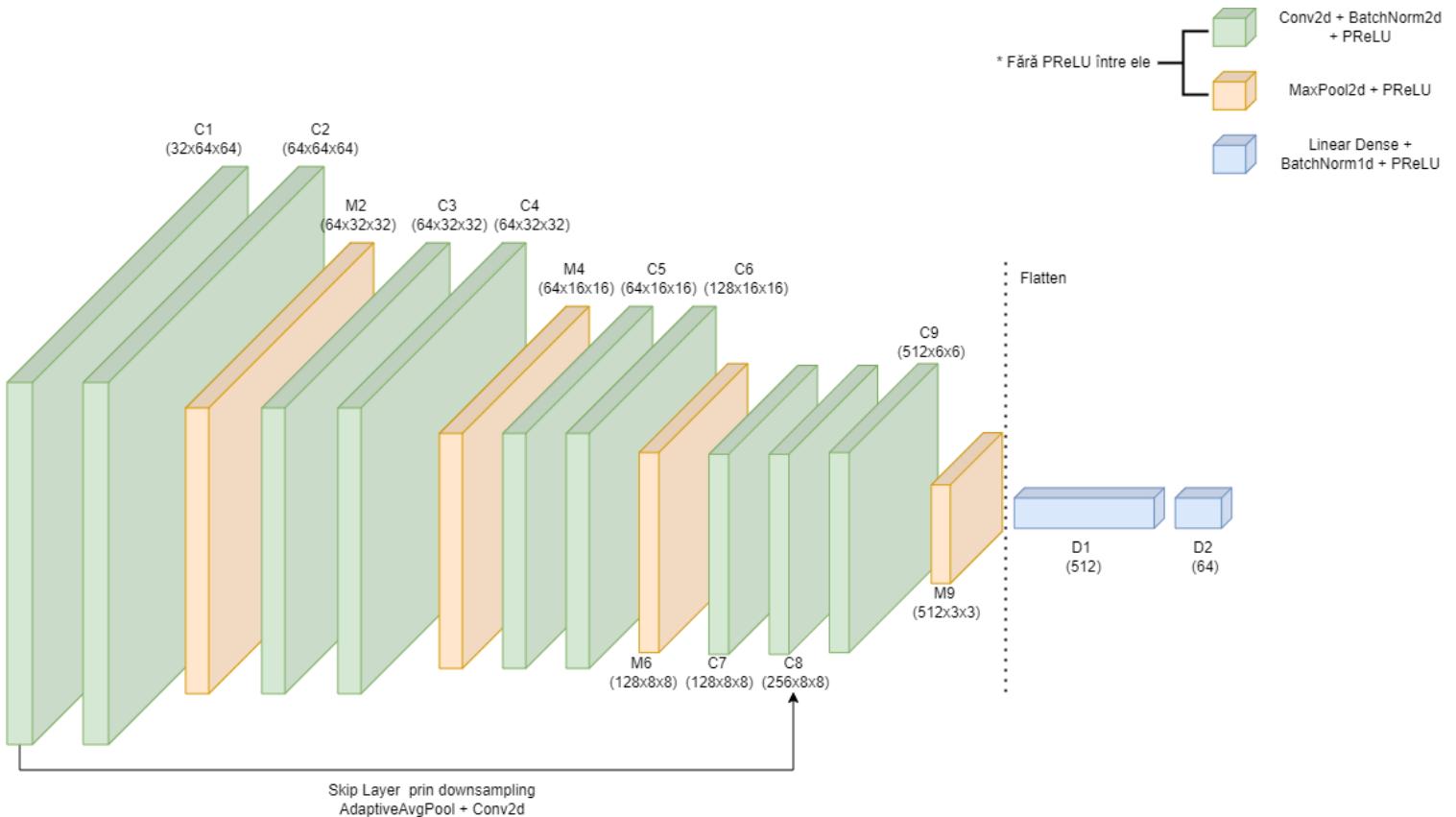
(0.943) **Test D:** Probabilitate 0.7 pe RandomVerticalFlip, 0.5 pe RandomHorizontalFlip

(0.9375) **Test E:** Probabilitate 0.5 pe RandomVerticalFlip, 0.5 pe RandomHorizontalFlip + RandomAffine(degrees=15, scale=(0.70, 1.30), shear=0.30)



➤ Concluzie CNN

➤ Model



Modelul final pentru CNN arată astfel:

C1-C8 sunt straturi conveționale cu batch normalization, `kernel_size=3, stride=1, padding=1`

C9 - strat convețional cu batch normalization, `kernel_size=5, stride=1, padding=1`

M2,M4,M6,M9 sunt straturi MaxPool2d cu `kernel_size=stride=2`.

D1,D2 sunt straturile dense/fully connected de după aplatizarea feature-urilor.

Pentru skip layer folosesc un AdaptiveAvgPool care preia dimensiunile lui x, și un Conv2d cu `kernel_size=3, stride=1, padding=0`, care potrivește numărul de channel-uri.

➤ Data Augmentation

Pentru îmbunătățirea antrenării imaginile suferă următoarele modificări:

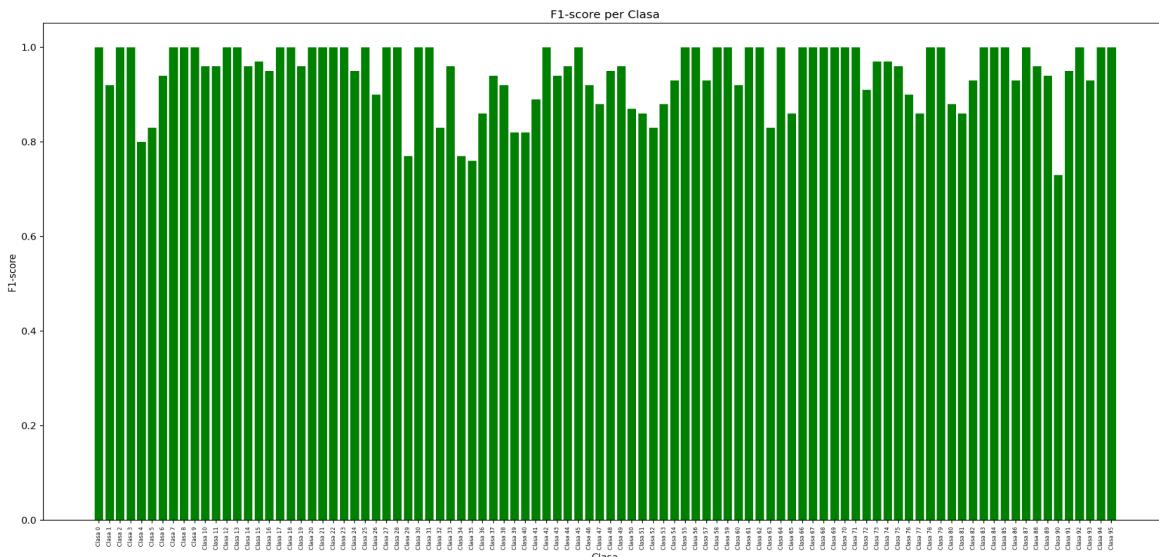
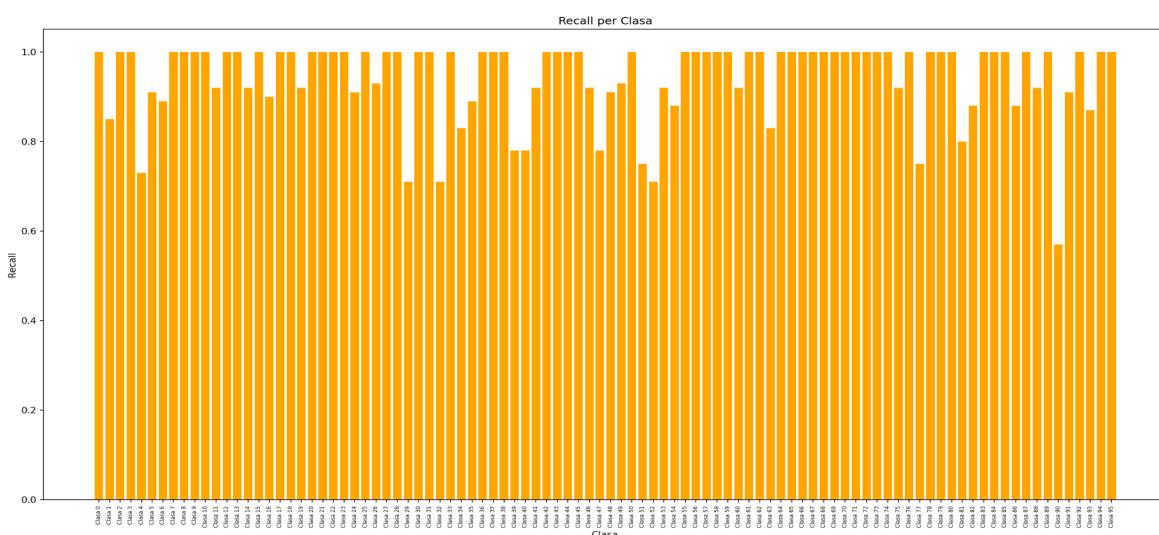
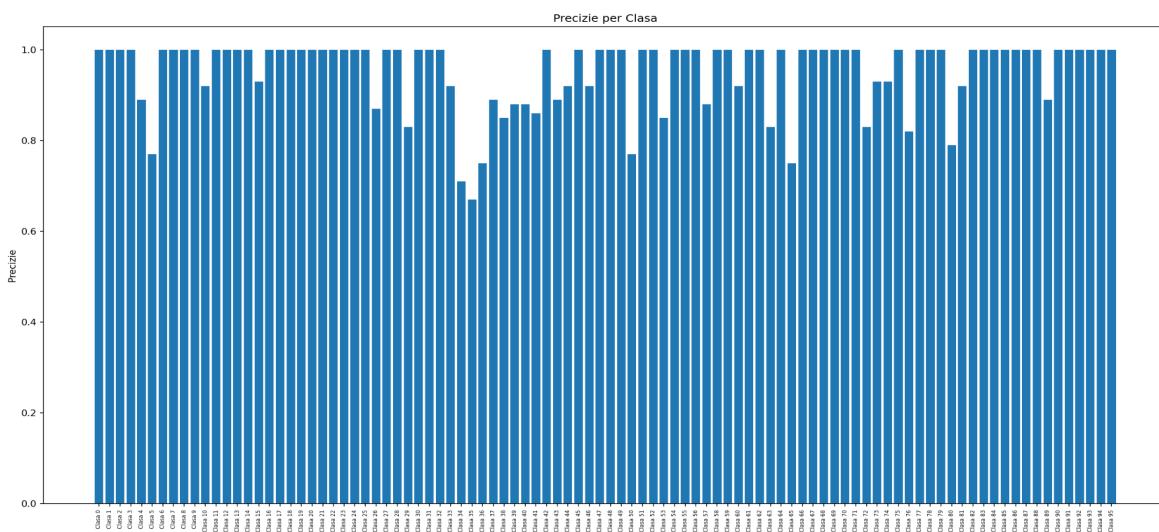
- pixelii sunt normalizați prin împărțire la 255
 - aplicăm transformări din pytorch asupra imaginilor, precum: RandomHorizontalFlip, RandomVerticalFlip, RandomErasing, RandomAffine, toate cu o probabilitate de 0.5 de a se aplica.

➤ Training Extras (Optimizer + Criterion + Scheduler + Early Stopping)

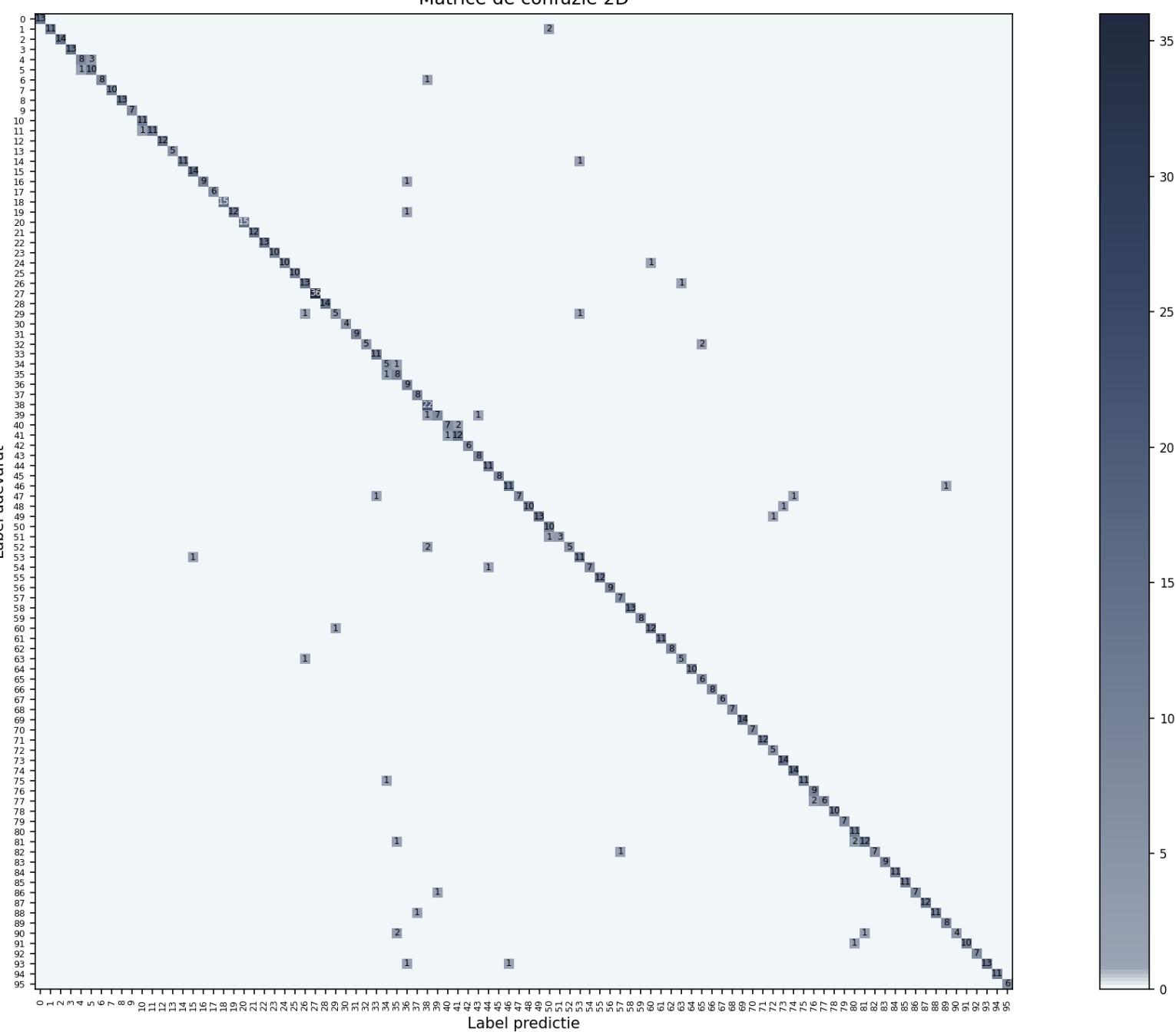
Pentru optimizer folosesc AdamW cu weight_decay=0.03, learning_rate=0.001, pentru criterion folosesc CrossEntropyLoss, pentru scheduler folosesc CosineAnnealingLR cu T_max numărul de epoci și eta_min=0.000013, și fac early stopping în funcție de acuratețea de validare, cu un patience de 30 de epoci.

➤ Fără date extra

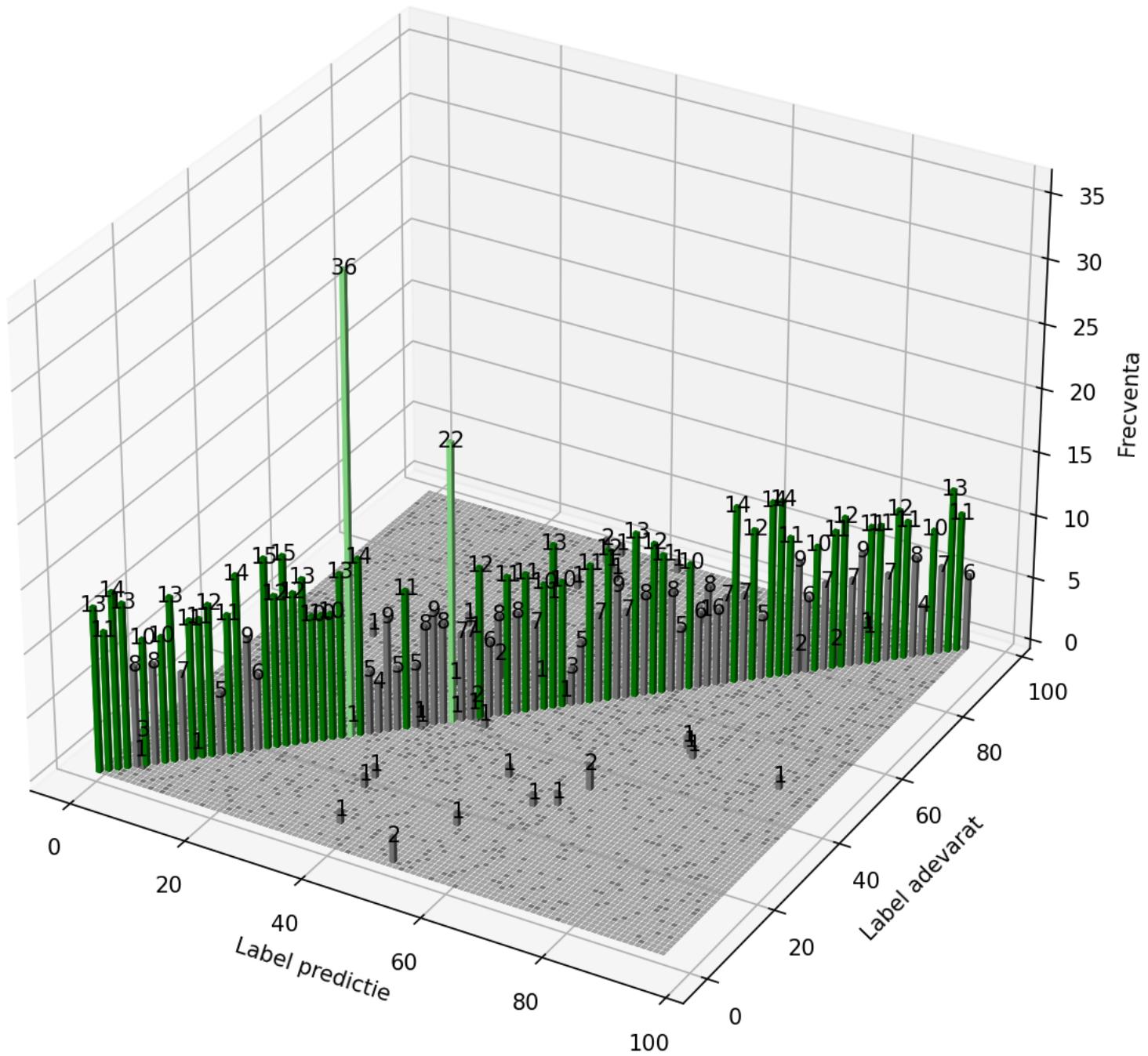
Surpriză la rulare, a dat o acuratețe mai bună decât cea anticipată. Diferențele dintre rulări sunt datorate factorilor imprevizibili precum shuffle-ul datelor de train, augmentarea imaginilor aleator.



Matrice de confuzie 2D



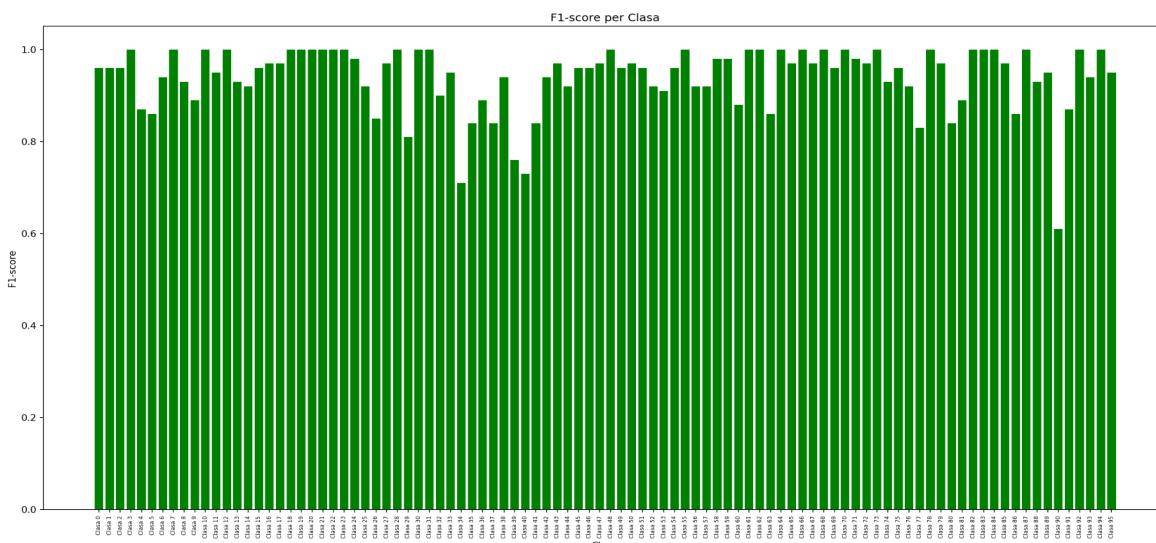
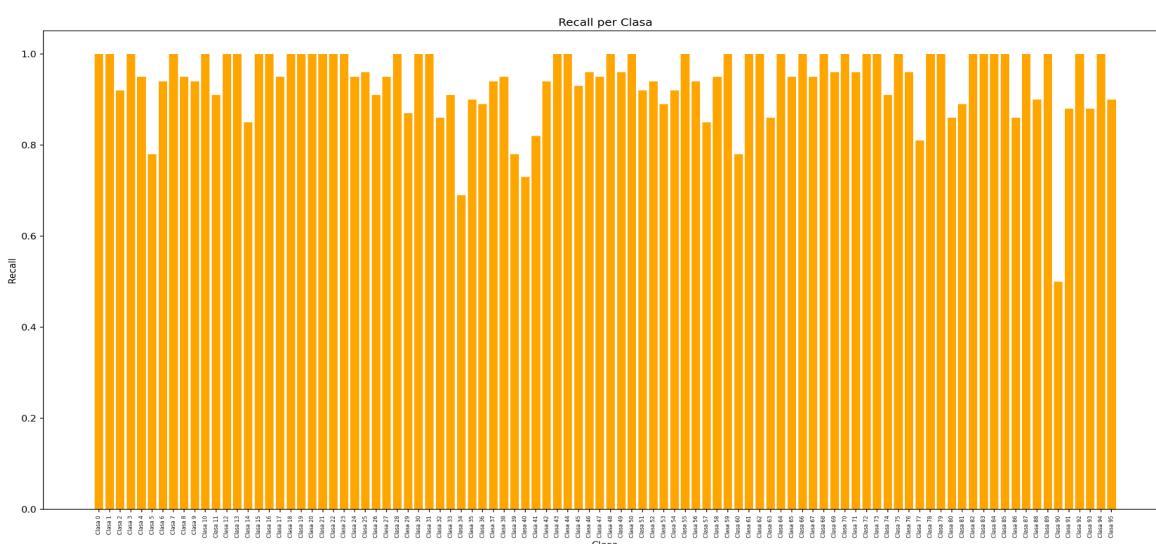
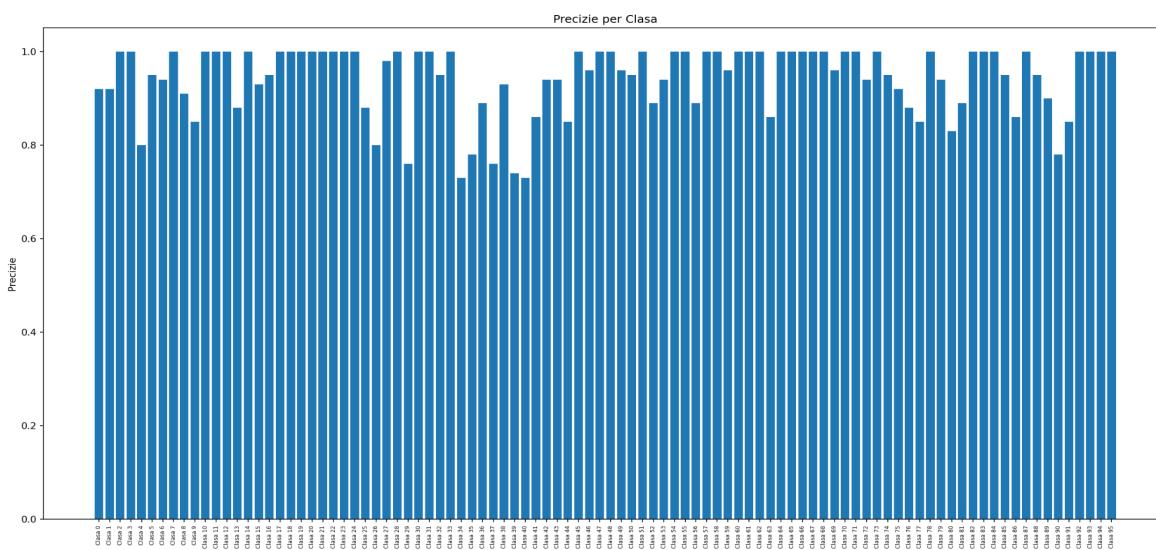
Matrice de confuzie 3D



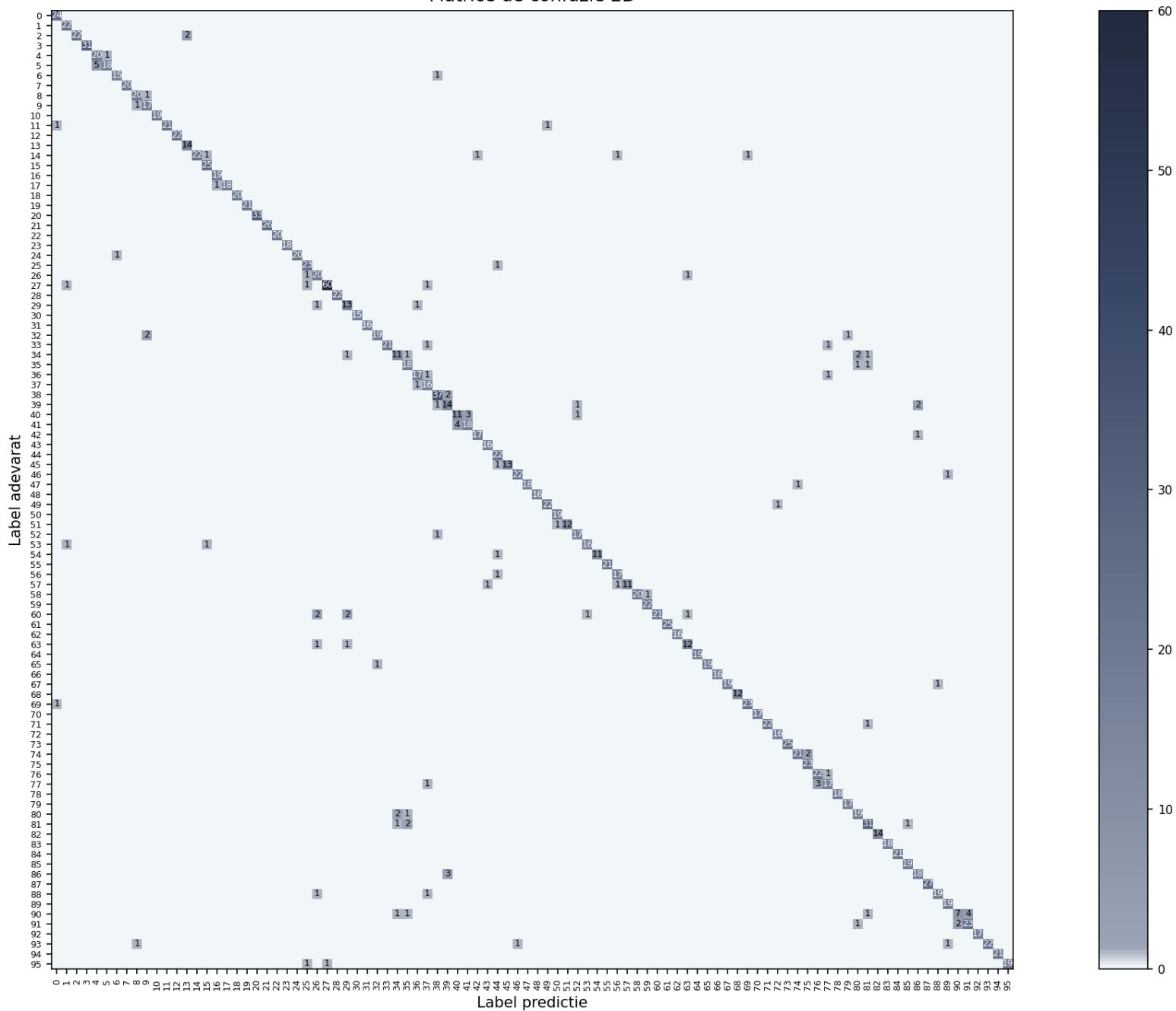
➤ Cu datele de la competiția anulată

	precision	recall	f1-score	support		Clasa 36	0.89	0.89	0.89	19		Clasa 75	0.92	1.00	0.96	23
Clasa 0	0.92	1.00	0.96	24		Clasa 37	0.76	0.94	0.84	17		Clasa 76	0.88	0.96	0.92	23
Clasa 1	0.92	1.00	0.96	22		Clasa 38	0.93	0.95	0.94	39		Clasa 77	0.85	0.81	0.83	21
Clasa 2	1.00	0.92	0.96	24		Clasa 39	0.74	0.78	0.76	18		Clasa 78	1.00	1.00	1.00	18
Clasa 3	1.00	1.00	1.00	31		Clasa 40	0.73	0.73	0.73	15		Clasa 79	0.94	1.00	0.97	17
Clasa 4	0.80	0.95	0.87	21		Clasa 41	0.86	0.82	0.84	22		Clasa 80	0.83	0.86	0.84	22
Clasa 5	0.95	0.78	0.86	23		Clasa 42	0.94	0.94	0.94	18		Clasa 81	0.89	0.89	0.89	35
Clasa 6	0.94	0.94	0.94	16		Clasa 43	0.94	1.00	0.97	16		Clasa 82	1.00	1.00	1.00	14
Clasa 7	1.00	1.00	1.00	20		Clasa 44	0.85	1.00	0.92	22		Clasa 83	1.00	1.00	1.00	18
Clasa 8	0.91	0.95	0.93	21		Clasa 45	1.00	0.93	0.96	14		Clasa 84	1.00	1.00	1.00	21
Clasa 9	0.85	0.94	0.89	18		Clasa 46	0.96	0.96	0.96	23		Clasa 85	0.95	1.00	0.97	19
Clasa 10	1.00	1.00	1.00	19		Clasa 47	1.00	0.95	0.97	19		Clasa 86	0.86	0.86	0.86	21
Clasa 11	1.00	0.91	0.95	23		Clasa 48	1.00	1.00	1.00	16		Clasa 87	1.00	1.00	1.00	27
Clasa 12	1.00	1.00	1.00	22		Clasa 49	0.96	0.96	0.96	23		Clasa 88	0.95	0.90	0.93	21
Clasa 13	0.88	1.00	0.93	14		Clasa 50	0.95	1.00	0.97	19		Clasa 89	0.90	1.00	0.95	19
Clasa 14	1.00	0.85	0.92	26		Clasa 51	1.00	0.92	0.96	13		Clasa 90	0.78	0.50	0.61	14
Clasa 15	0.93	1.00	0.96	25		Clasa 52	0.89	0.94	0.92	18		Clasa 91	0.85	0.88	0.87	26
Clasa 16	0.95	1.00	0.97	19		Clasa 53	0.94	0.89	0.91	18		Clasa 92	1.00	1.00	1.00	17
Clasa 17	1.00	0.95	0.97	19		Clasa 54	1.00	0.92	0.96	12		Clasa 93	1.00	0.88	0.94	25
Clasa 18	1.00	1.00	1.00	20		Clasa 55	1.00	1.00	1.00	21		Clasa 94	1.00	1.00	1.00	21
Clasa 19	1.00	1.00	1.00	21		Clasa 56	0.89	0.94	0.92	18		Clasa 95	1.00	0.90	0.95	21
Clasa 20	1.00	1.00	1.00	33		Clasa 57	1.00	0.85	0.92	13						
Clasa 21	1.00	1.00	1.00	26		Clasa 58	1.00	0.95	0.98	21						
Clasa 22	1.00	1.00	1.00	20		Clasa 59	0.96	1.00	0.98	22						
Clasa 23	1.00	1.00	1.00	18		Clasa 60	1.00	0.78	0.88	27						
Clasa 24	1.00	0.95	0.98	21		Clasa 61	1.00	1.00	1.00	25						
Clasa 25	0.88	0.96	0.92	24		Clasa 62	1.00	1.00	1.00	16						
Clasa 26	0.80	0.91	0.85	22		Clasa 63	0.86	0.86	0.86	14						
Clasa 27	0.98	0.95	0.97	63		Clasa 64	1.00	1.00	1.00	19						
Clasa 28	1.00	1.00	1.00	22		Clasa 65	1.00	0.95	0.97	20						
Clasa 29	0.76	0.87	0.81	15		Clasa 66	1.00	1.00	1.00	16						
Clasa 30	1.00	1.00	1.00	15		Clasa 67	1.00	0.95	0.97	20						
Clasa 31	1.00	1.00	1.00	16		Clasa 68	1.00	1.00	1.00	12						
Clasa 32	0.95	0.86	0.90	22		Clasa 69	0.96	0.96	0.96	24						
Clasa 33	1.00	0.91	0.95	23		Clasa 70	1.00	1.00	1.00	17						
Clasa 34	0.73	0.69	0.71	16		Clasa 71	1.00	0.96	0.98	23						
Clasa 35	0.78	0.90	0.84	20		Clasa 72	0.94	1.00	0.97	16						
						Clasa 73	1.00	1.00	1.00	25						
						Clasa 74	0.95	0.91	0.93	23						

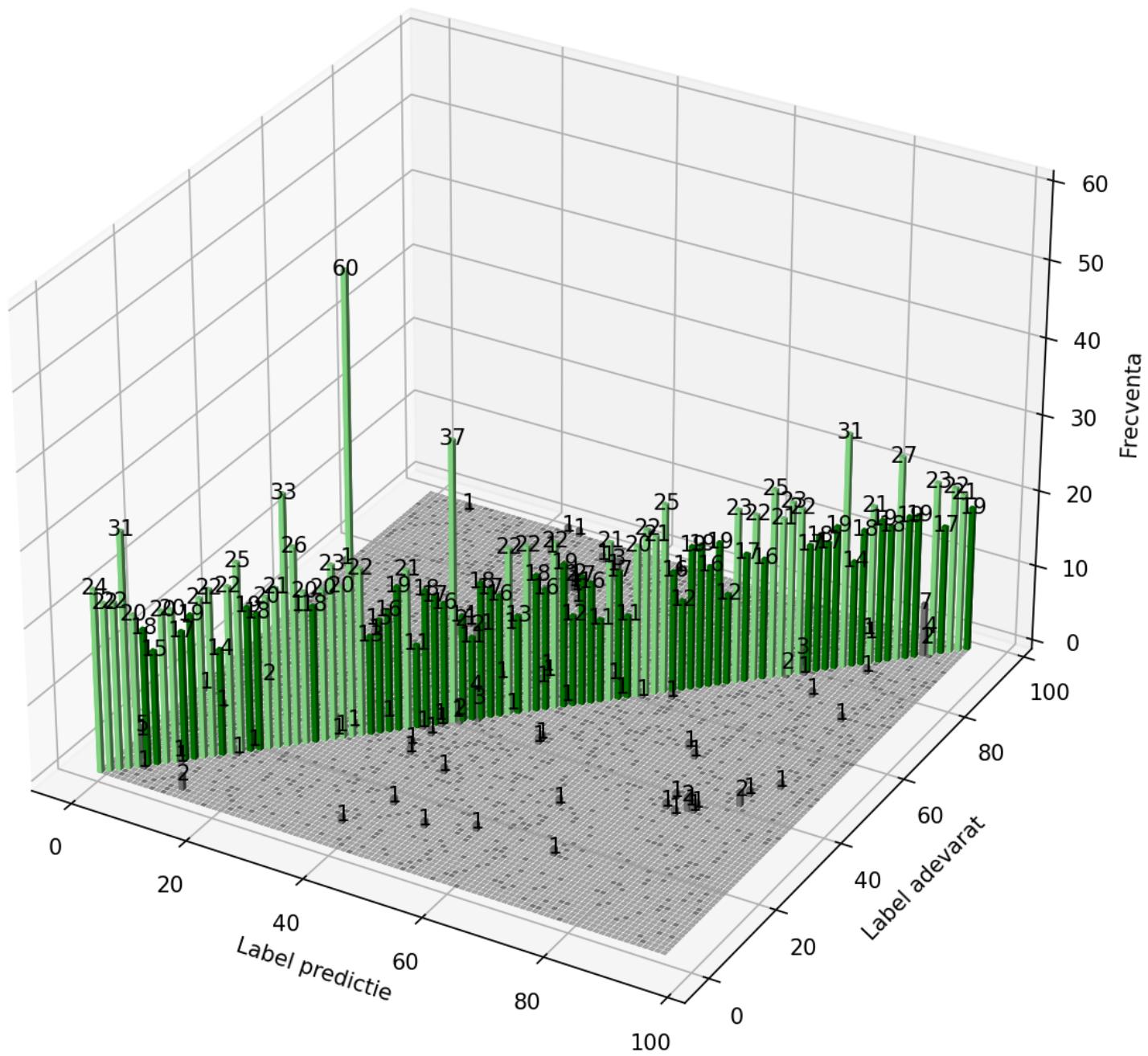
Deși acuratețea nu este vizibil mai mare decât în cazul fără datele de la competiția anulată, trebuie ținut cont că sunt 2000 de date de validare în această varianta, și overall generalizează mai bine și obține un scor mai bun pe Kaggle deoarece are mai multe de învățat.



Matrice de confuzie 2D



Matrice de confuzie 3D



➤ O ultimă comparație între cele două rulări de mai sus

