CNN

Pentru a rezolva cerinta, am ales ca abordare principala sa implementez o retea neuronala convolutionala, aceasta fiind ideala pentru tipul de date oferit.

In prima faza, am implementat eu anumite functii, cum ar fi convolve, max pool, relu, softmax si CrossEntropyLoss si am incercat sa fac un CNN simplificat bazat pe ele, ca sa compar eficienta si sa am o reprezentare vizuala. Cum codul rula extrem de lent, am continuat folosind functiile implementate in PyTorch.

0. Prima incercare de CNN, fara augmentari pentru imaginile de training, cu un singur strat fully connected.

Structura:

- 3 straturi convolutionale 3x3 si padding 1
- relu si maxpool dupa fiecare
- un singur strat fully connected
- functia de pierdere: CrossEntropyLoss()
- optimizator: Adam

Parametri:

- batch size: 32
- learning rate: 0.01
- epoci: 35

Augmentari:

- nu am folosit la aceasta incercare
- doar normalizare standard

Rezultat: 75% accuracy pe datele de validare

Aici nu am facut grafic/matrice de confuzie. Pentru a imbunatati rezultatul, la urmatoarea incercare am implementat un CNN cu o structura mai complexa si am folosit augmentari pe imaginile de training.

1.

Structura:

- 3 straturi convolutionale 3x3 si padding 1
- relu si maxpool dupa fiecare

- dropout cu probabilitate 0.4 dupa ultimul strat convolutional si dupa primul strat fully-connected
- 2 straturi fully connected
- CrossEntropyLoss
- Adam

Parametri:

- batch size: 32
- learning rate: 0.001
- epoci: 35

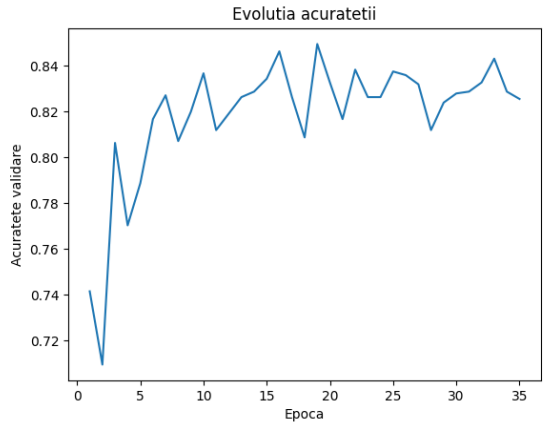
Augmentari:

- rotatie aleatoare ±10°
- flip orizontal aleatoriu

Rezultate:

- 84.96% accuracy pe datele de validare
- scor pe Kaggle: 0.8633

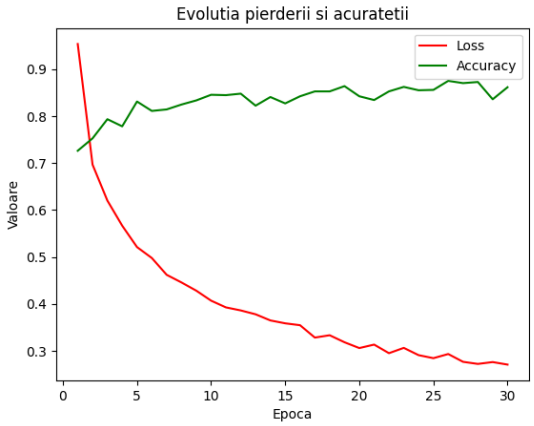
Augmentarile au dus la o generalizare mai buna a modelului. In plus, a fost redusa suprainvatarea, datorita dropout-ului si ratei de invatare.

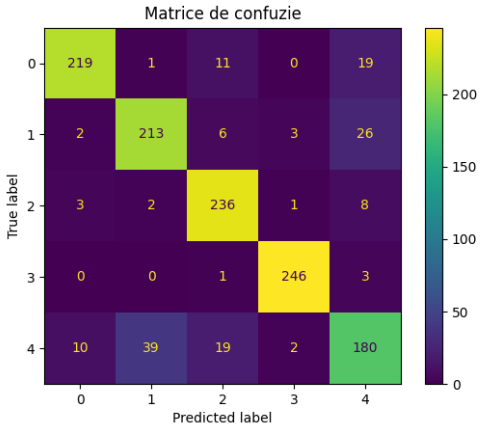


2. Aici structura este identica, doar am scazut numarul de epoci la 30 si am crescut dropout-ul la 0.5.

Rezultate:

- 87.43% accuracy pe datele de validare
- scor pe Kaggle: 0.868





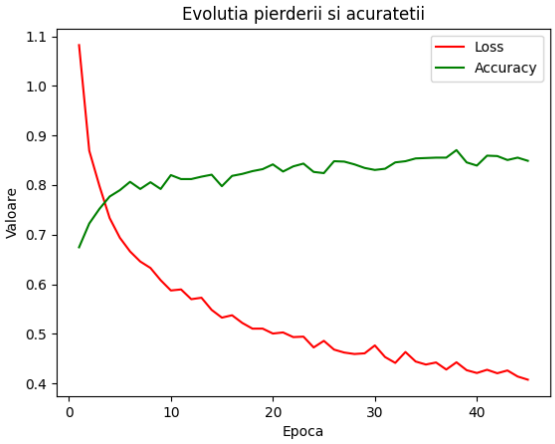
3. Am adaugat la augmentari:

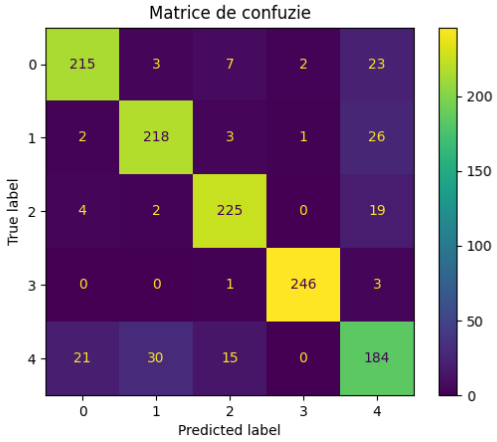
transforms.ColorJitter(brightness=0.25, contrast=0.15, saturation=0.2, hue=0.1),
transforms.RandomAffine(degrees=0, translate=(0.05, 0.05), scale=(0.9, 1.1)),

Epoci: 45

Rezultate:

- 87.52% accuracy pe datele de validare
- scor pe Kaggle: 0.864





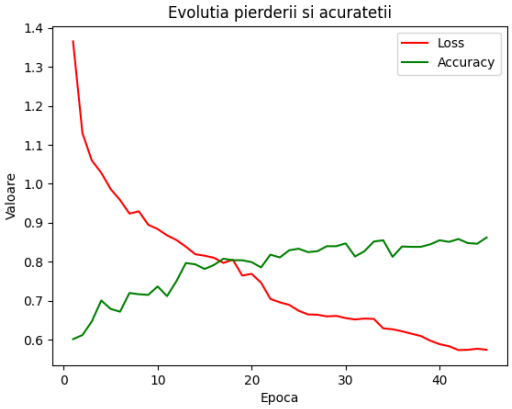
4. Am remarcat din matricea de confuzie obtinuta in rularile anterioare ca modelul greseste mai mult in cazul imaginilor din clasa 4, asa ca am incercat sa ii pun acesteia o pondere mai mare in functia de pierdere, ca sa fie penalizate mai puternic greselile. Pentru asta, am facut un vector de frecvente in care am pus 250 pentru clasele 0-3 si 180 pentru 4 , astfel ca pondere acesteia in functia de pierdere era 0.005, spre deosebire de 0.004 pentru celelalte. In aceasta rulare nu s-a observat o diferenta drastica, dar in rulari ulterioare, in care aveam mai multe epoci si o arhitectura mai complexa, greselile de la clasa 4, chiar daca in continuare majoritare, s-au apropiat de cele de la celelalte clase.

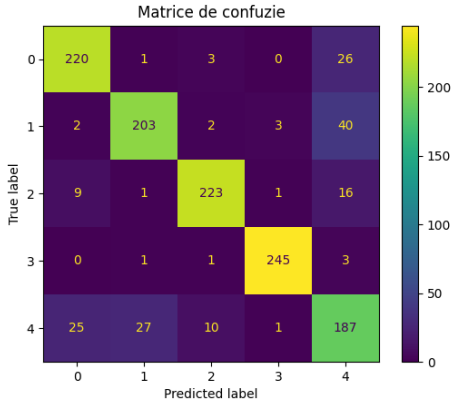
In plus, am adaugat batch normalisation dupa fiecare convolutie si un scheduler pentru learning rate, care sa o injumatateasca daca acurateatea stagneaza 3 epoci.

Dropout: 0.45

Rezultate:

- 87.8% accuracy pe datele de validare
- scor pe Kaggle: 0.861

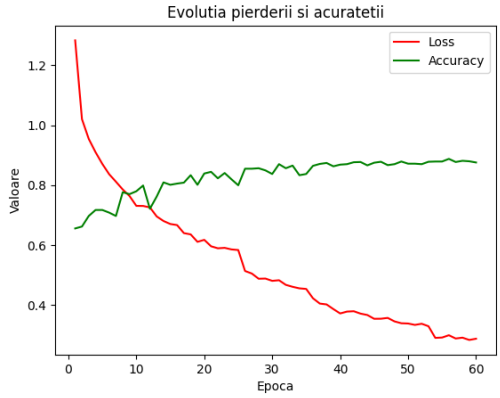


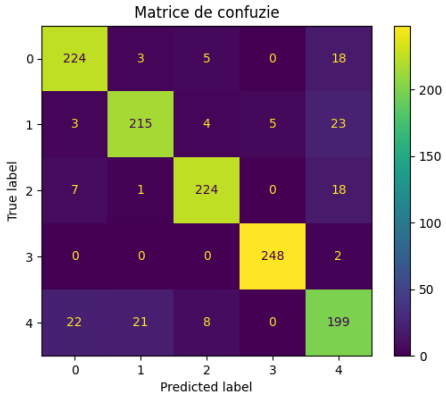


5. Aici am incercat in continuare sa reduc suprainvatarea, asa ca am adaugat inca un strat fully connected, weight decay in optimizer si am crescut numarul de epoci.

Dropout: 0.45

Epoci: 60





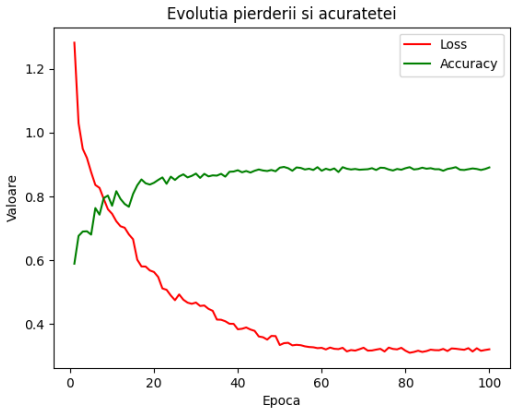
Rezultate:

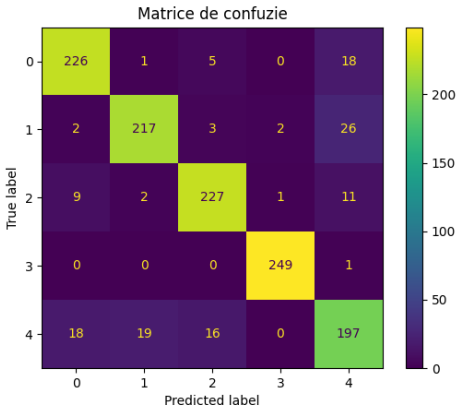
- 88.8% accuracy pe datele de validare
- scor pe Kaggle: 0.897

6. Aici doar am trecut antrenarea pe GPU pentru a creste viteza si am crescut numarul de epoci la 100.

Rezultate:

- 89.28% accuracy pe datele de validare
- scor pe Kaggle: 0.897





7. Pentru aceasta incercare am adaugat inca un strat convolutional, am scazut dropout-ul si am inlocuit optimizatorul Adam cu AdamW, pentru a imbunatati regularizarea.

In ce priveste augmentarile, am crescut imaginile de la 100x100 la 160x160, pentru a ramane cu mai multe informatii dupa max pool.

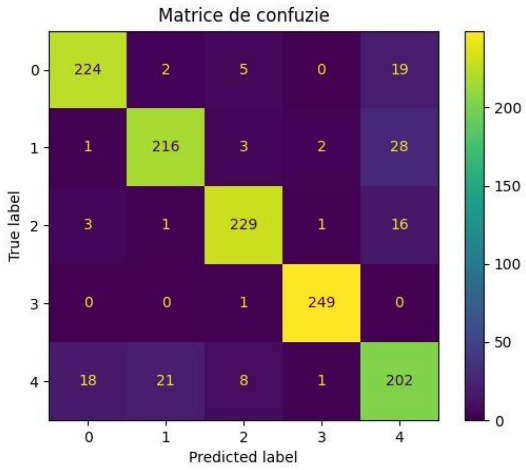
Epoci: 100

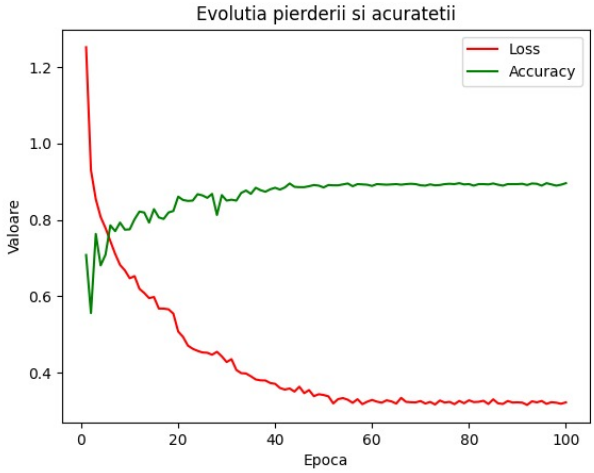
Dropout: 0.35

Learning rate: 0.0002

Rezultate:

- 90.28% accuracy pe datele de validare
- scor pe Kaggle: 0.904



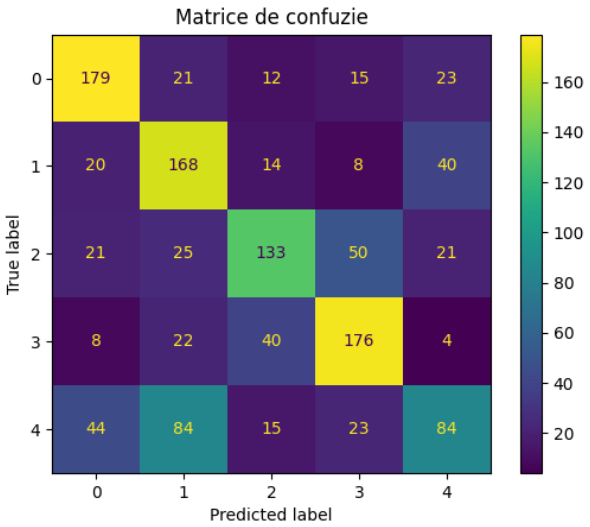


SVM:

Al doilea model pe care l-am implementat pentru a rezolva tema a fost un SVM cu histograme de culoare, cu un kernel linear(LinearSVC). Aceasta abordare a fost utila pentru a compara diferentele dintre doua modele pe acelasi task.

Modelul a atins o acuratete de 63% pe imaginile de validare, similar cu ce atingea un CNN dupa prima epoca, in aproximativ acelasi timp. In matricea de confuzie am remarcat ca predictiile acestui model au avut tendinte similar cu cele din CNN, cum ar fi greselile extreme(daca true label e 4 si modelul greseste, de cele mai multe ori prezice 0 sau 1, nu 2 sau 3) si faptul ca majoritatea greselilor au loc la imagini din clasa 4.

Consider ca performanta slaba este o consecinta a faptului ca modelul se bazeaza pe distributia culorilor in imagine(care se realizeaza in cod pe 32 de intervale), nu pe trasaturi mai specifice, astfel ca mai multe imagini ajung sa aiba histograme foarte asemanatoare, chiar daca au continut diferit.



Bibliografie:

Curs/Laborator

https://medium.com/thecyphy/train-cnn-model-with-pytorch-21dafb918f48

https://www.kaggle.com/code/eugeniyosetrov/using-data-augmentation-for-cnn

https://docs.pytorch.org/vision/0.9/transforms.html

https://gurjeet333.medium.com/7-best-techniques-to-improve-the-accuracy-of-cnn-w-o-overfitting-6db06467182f

https://www.geeksforgeeks.org/adam-optimizer/

https://docs.pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

https://docs.pytorch.org/docs/stable/generated/torch.optim.AdamW.html

https://www.ibm.com/think/topics/support-vector-machine#:~:text=How%20SVMs%20work-,What%20are%20SVMs%3F,in%20an%20N%2Ddimensional%20space.