# *DETECȚIE MĂȘTI*

Setul de date: Datele sunt impartite in 3 clase, cu numar inegal de poze per clasa si se gaseste la link-ul : https://www.kaggle.com/datasets/andrewmvd/face-mask-detection Acestea au fost preluate intr-un format ideal, au fost create special pentru a antrena un model, astfel nu au fost necesare multe preprocesari, doar verificari legate de dimensiuni. Modelul clasifica persoanele care poartă măști, nu le poartă sau le poartă în mod necorespunzător.

Am testat/antrenat mai multe modele, pentru fiecare am calculat acuratetea,train/validation loss.

*Model preantrenat - CNNWithResNet*

Bază: Utilizează ResNet-18 de la PyTorch, adaptat pentru clasificare specifică prin înlocuirea ultimului strat.

Activare: Folosește implicit softmax prin pierderea entropiei încrucișate în antrenament.

Funcție de Pierdere : Entropie Încrucișată, optimă pentru clasificarea multi-clasă.

Optimizator: Adam, adecvat pentru gestionarea gradienților și adaptarea ratei de învățare.

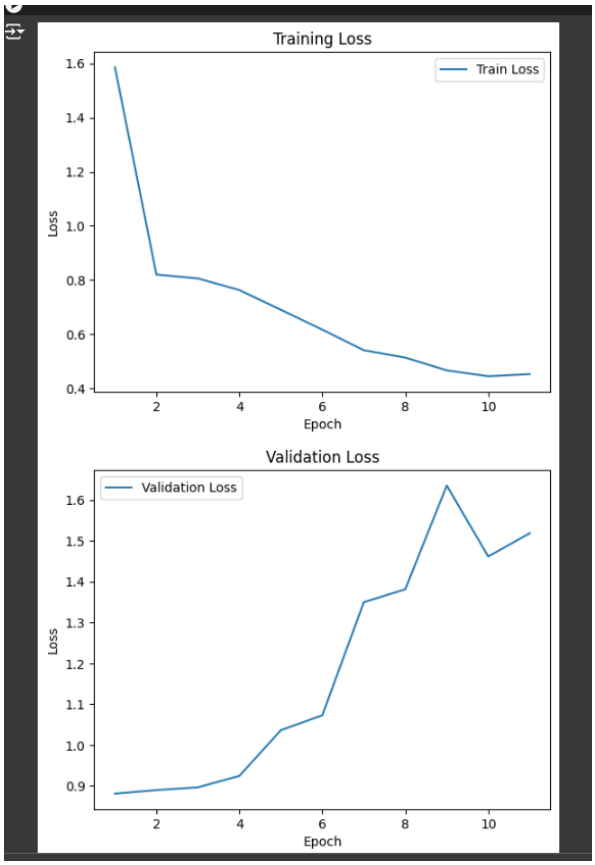
Regim de Antrenament: Epoci: 50, cu implementarea opririi timpurii după 3 epoci fără îmbunătățiri. Performanță:

Monitorizare: Urmărește acuratețea de validare pentru a evalua generalizarea modelului.

Observații: Eficiență: Antrenamentul reutilizează majoritatea arhitecturii ResNet-18, fiind mai eficient din punct de vedere computațional.

*Model creat de mine - SimpleCNN*

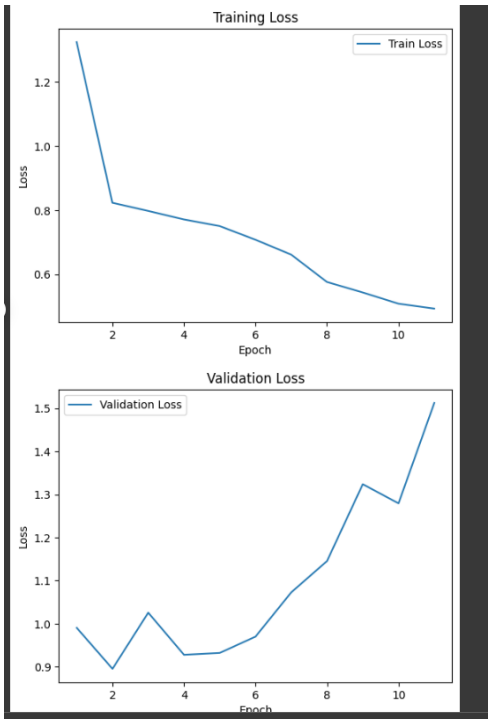
Modelul meu, SimpleCNN, utilizează o arhitectură cu trei straturi convoluționale, fiecare urmat de ReLU și MaxPooling, crescând numărul de canale de la 3 la 128. Aplatizarea output-ului convoluțional alimentează două straturi fully connected. Folosești CrossEntropyLoss și optimizerul Adam pentru a antrena modelul pe 50 de epoci, structurând antrenamentul și validarea pe seturi împărțite ale datelor. Observatii: modelul se antreneaza decent, este simplu structurat.



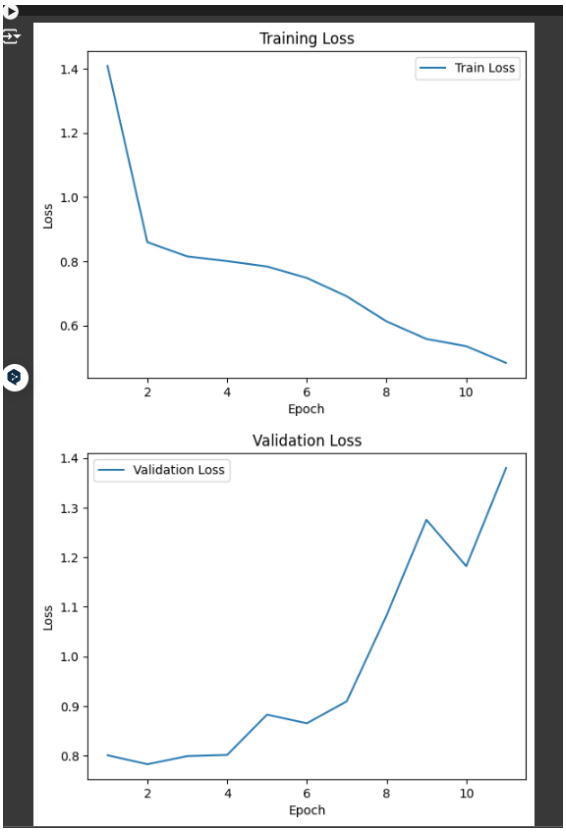

Experimente cu batch size-ul

Cu batch size mai mic, modelul preia mai in amanunt informatiile, ceea ce face ca trainingul sa fie mai lung.

- mic

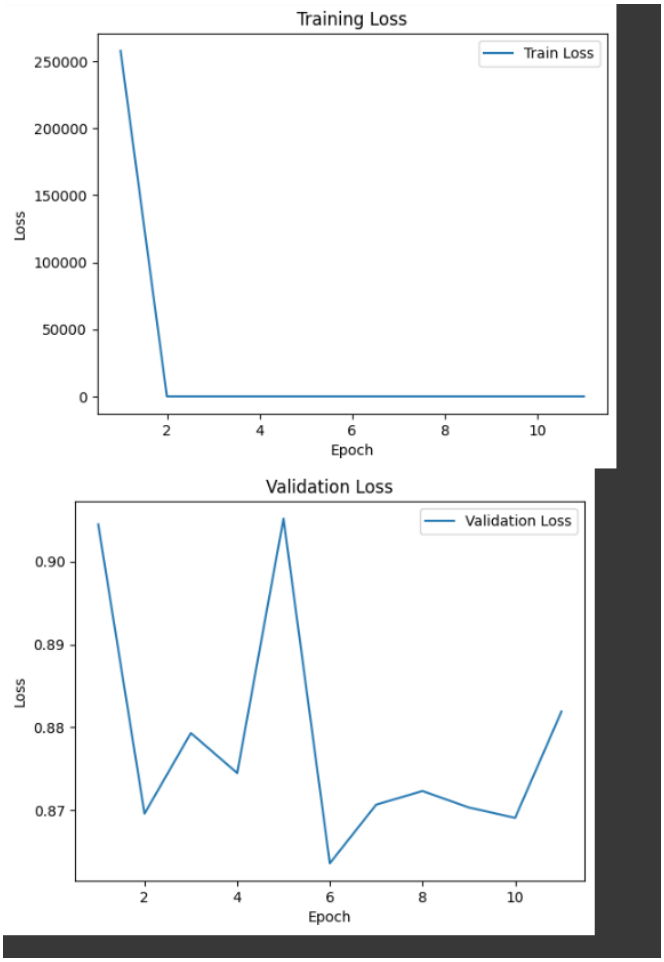

Training Loss
Train Loss
Loss
Epoch
Validation Loss
Validation Loss
Loss
Epoch

- mare


Training Loss
Train Loss
Loss
Epoch
Validation Loss
Validation Loss
Loss
Epoch

Experimente cu learning rate-ul

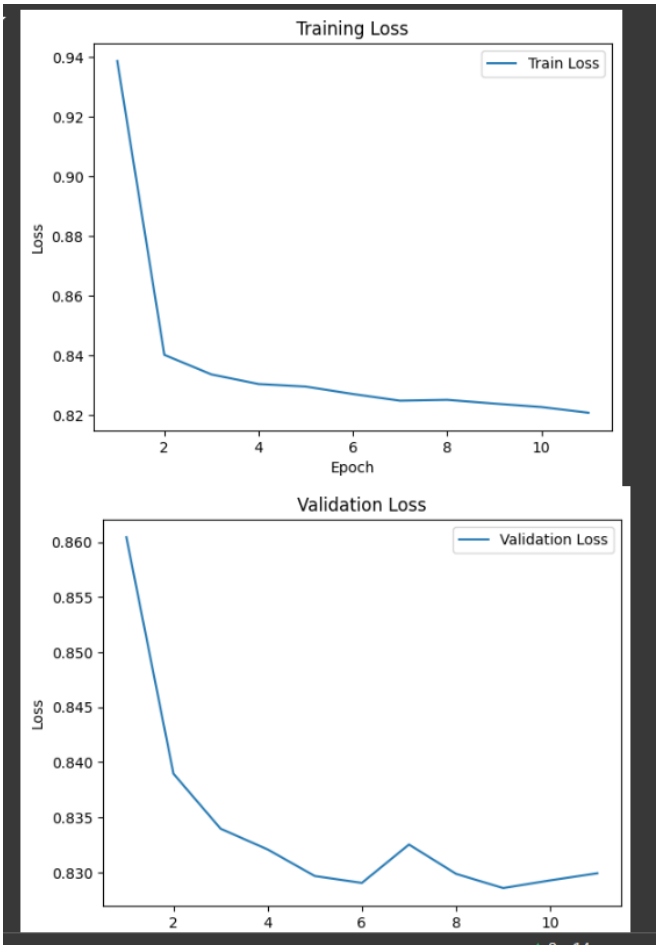
Ca la Observatii generale doar ca la Observatii: modelul se antreneaza bine, este simplu structurat si are performante bune. Scheduler(optimieaza-update la parametrii dupa fiecare epoca) il am prezent in Bulina 7.




Experimente cu optimizatorii

Ca la Observatii generale doar ca la Optimizator avem SGD in loc de Adam.

Atât SGD, cât și Adam sunt algoritmi de optimizare foarte populari în contextul clasificării. Diferența principală între cei doi constă în faptul că SGD utilizează un learning rate constant, în timp ce Adam are un learning rate variabil. Deși modelul optimizat cu SGD are o performanță bună, în acest caz particular, s-a observat că performanța este inferioară comparativ cu modelul optimizat cu Adam. Cu toate acestea, pentru clasificarea de imagini, se pare că SGD ar fi mai potrivit datorită stabilității oferite de learning rate-ul constant, care poate facilita convergența către soluția optimă în anumite scenarii.




Function losses

Ca la Observatii generale doar ca la Observatii: modelul se antreneaza bine, este simplu structurat si are performante bune.

Am facut o ponderata/combinatie pe functiile de loss folosite.

Am folosit scheduler aici in loc de Bulina 5.

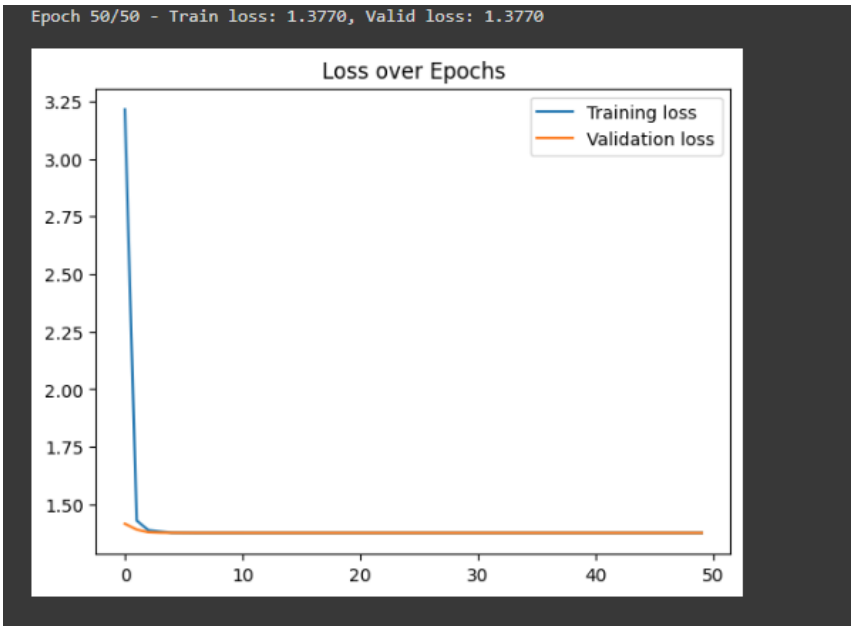
Sigur, iată explicațiile pentru fiecare criteriu de pierdere în 5 rânduri:

CrossEntropyLoss este utilizată pentru sarcinile de clasificare, calculând pierderea între probabilitățile prezise ale claselor și etichetele reale. Combină nn.LogSoftmax() și nn.NLLLoss(), facilitând procesul de antrenare. Se așteaptă ca etichetele țintă să fie indici de clasă. Este ideală pentru clasificare multi-clasă și necesită ca stratul de ieșire să aibă o activare softmax.

Observatii: Cam aceas problema ca la Relu, exponentialul de la un punct se plafoneaza si nu ofera o distributie foarte buna pentru clasificatorul modelat de mine. In practica este foarte util, dar nu se modeleaza bine pe structura creata de mine.

KLDivLoss măsoară divergența Kullback-Leibler între două distribuții probabilistice, evaluând cât de mult diferă o distribuție de la alta. Este folosită frecvent în modele probabilistice, precum Variational Autoencoders (VAEs) sau distilarea modelului profesor-elev. Parametrul reduction='batchmean` asigură media pierderii pe lot. Intrarea trebuie să fie log-probabilități, iar ținta probabilități.

MSELoss calculează eroarea pătratică medie între valorile prezise și cele reale, fiind folosită în sarcinile de regresie. Este ideală pentru probleme unde ieșirea este o variabilă continuă. Aceasta ajută la minimizarea diferențelor pătratice dintre predicții și ținte. Atât intrarea, cât și ținta trebuie să aibă aceeași formă.




OPT, Accuracy Recall, Confusion Matrix

E destinat să ofere predicții mai robuste și să evalueze incertitudinea acestora prin entropie. În primul pas, un model antrenat este adăugat la o listă de modele de ansamblu. Următorul pas implică inițializarea indicilor pentru seturile de antrenament și validare și crearea de sampler-e care extrag aleatoriu exemple din aceste seturi, utilizate apoi pentru a alimenta încărcătoarele de date.

Funcția de entropie este definită pentru a calcula incertitudinea predicțiilor, folosind funcția softmax pentru a estima probabilitățile, după care se aplică logaritmii acestora pentru a obține entropia. În procesul de predicție, pentru fiecare lot de imagini din setul de validare, fiecare model din ansamblu contribuie la predicția finală, iar rezultatul este mediat pentru a forma predicția de ansamblu.

Dacă apar erori în timpul predicției, sunt afișate mesaje de eroare și se sugerează pași pentru remedierea situației. Când se obțin predicțiile de ansamblu, se calculează entropia acestora ca măsură a incertitudinii. De asemenea, se evaluează performanța modelului prin calculul acurateței și recall-ului și se generează o matrice de confuzie pentru a vizualiza performanța clasificărilor.

În final, rezultatele și analizele sunt integrate în TensorBoard prin intermediul unui SummaryWriter din TensorBoard, permițând vizualizarea și analiza mai detaliată a performanței modelului de ansamblu. Această implementare ilustrează cum ansamblurile de modele pot îmbunătăți fiabilitatea predicțiilor și oferi evaluări ale incertitudinii în aplicații de învățare automată.

Cross validation

Începe cu inițializarea variabilelor cheie pentru acuratețe, pacientă și numărul de epoci. Folosind KFold, împarte setul de date în subseturi pentru antrenament și validare. Un model, criteriul de pierdere și un optimizer sunt configurate pentru fiecare fold. În fiecare epocă, datele sunt procesate în batch-uri pentru antrenament, iar apoi modelul este evaluat pe setul de validare. Acuratețea este calculată și comparată cu cea mai bună acuratețe anterioară, iar antrenamentul poate fi oprit dacă nu există îmbunătățiri după un număr definit de epoci. La sfârșit, sunt afișate rezultatele pentru fiecare fold.