Documentie Proiect Machine Learning

Pascaru Dan-Alexandru Gr 251

1. Introducere

Tema proiectului a fost clasificare corecta in 5 clase a unor imagini deepfake. Setul de date de antrenament contine 12500 de imagini, cel de validare 1500 iar cel de testare 6500. Pentru ca primim si label-urile pentru imaginile de antrenament si validare am ales sa folosesc algoritmi de antrenare supervizata: Suport Vector Machine (SVM) si Convolutional Neural Network (CNN).

Antrenarea modelelor a fost realizata pe calculatorul personal : Ryzen 9 7900x, 64 GB RAM si RTX 4070.

2. CNN

Primul model pe care l-am ales se bazeaza pe o retea neuronala convolutionala. Acest tip de retele se caracterizeaza prin folosirea diferitelor tipuri de filtre pentru a extrage trasaturi reprezentative care contribuie la generalizarea unor sabloane. In prima partea a proiectul pentru CNN am utilizat Keras pentru ca am crezut ca o sa fie mai simplu si intuitiv, dar am facut trecerea la Pytorch deoareace ofera mai multe functionalitati avansate deja implementate si optimizate.

Modelul initial a fost construit cu o structura simpla formata din trei straturi convolutionale cu 64, 128 si 256 de filtre, toate cu kerneluri de dimensiune 3x3. Am folosit functia de activare ReLU, pentru ca ni s-a spus la curs ca este o functie standard care se comporta foarte bine. Pentru regularizare am aplicat un Dropout de 0.5 intre primele doua straturi. Dupa straturile convolutionale, am aplicat Flatten pentru a transforma rezultatul intr-un vector unidimensional urmat de un strat de activare Softmax care extrage probabilitatile fiecarei clase de apartenenta.

Pentru optimizare am folosit algoritmul Adam cu o rata de invatare de 0.001, deoarece foloseste momentum-ul in mod eficient pentru adaptarea gradientului. Ca functie de pierdere am ales sparse_categorical_crossentropy, considerata mai eficienta din punct de vedere al memoriei decat categorical_crossentropy, dar comportandu-se identic. Primele rezultate obtinute au fost 72.13% dupa 30 de epoci si 76.13% dupa 60 de epoci.

Prelucrarea datelor a avut un impact major asupra performantelor obtinute. Initial nu am aplicat nicio forma de normalizare sau augmentare. Am introdus tehnici precum: normalizarea (medie si varianta), rotatii (90 si 270 de grade), crop-urile aferente celor 4 colturi, zoom, luminozitate si contrast. Deoarece aveam 100000 de date, am ajustat modelul arhitectural crescand capacitatea retelei: 64x64, 128x128, 256x256, 512x512, fiecare cu BatchNormalization, MaxPooling si Dropout

0.5, iar toate kernelurile de 3x3 mai putin pe stratul de 64x64 unde am pus 5x5, apoi am scazut rata de invatare la 0.0001. Aceste modificari ne-au obtinut acuratete de 90-91%.

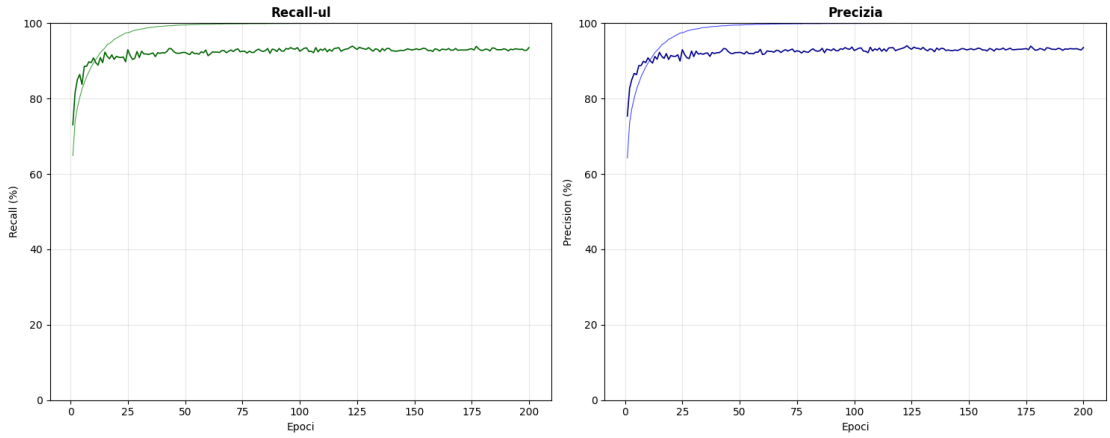
Conversia imaginilor in alb-negru a avut un efect negativ, scazand acuratetea modelului la 83%-84%. Am incercat atat antrenarea pe alb-negru a augumentarilor de mai sus, cat si cu imagini colore, dar amestecarea lor a dus modelul la 78%-80%. Asadar am renuntat la aceasta augumentare.

In general am antrenat cam 200-250 de epoci dar am lasat modelul sa se antreneze 750 de epoci (aproximativ 19 ore) si am reusit sa obtinem o varianta a modelului care era constant intre 90% si 92%. Am adaugat si antrenarea random a datelor si gruparea in 64 de batch-uri, precum si salvarea celor mai bune 3 modele dupa rata de evaluare. Am reusit sa obtinem 92.5%.

Observand ca odata cu scaderea ratei de invatare modelul a inceput sa devina mai stabil, am abordat o strategie noua bazata pe Cosine Annealing: $\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{T_{\text{cur}}}{T_{\max}}\pi\right)\right)$. Scaderea dinamica a ratei de invatare de la 0.001 la 0.00001 a imbunatatit rezultatele intre 92.1% si 92.8% acuratete de evaluare, iar cele mai bune modelele au obtinut 92.533%, respectiv 92.800%.

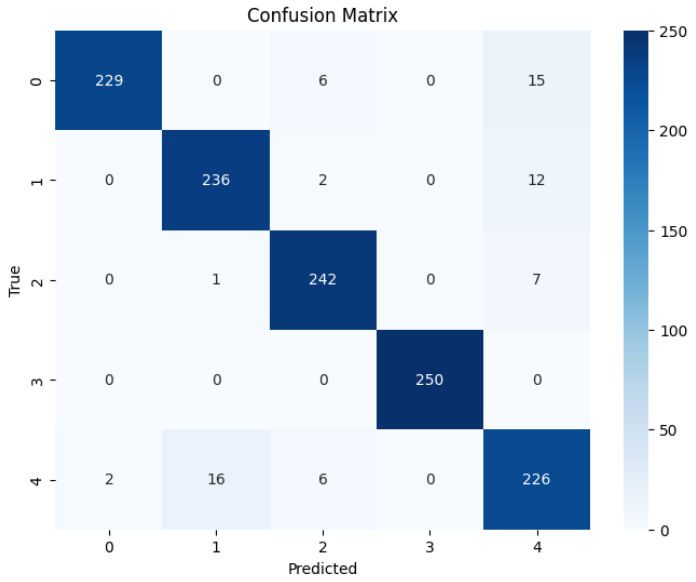
Avand in vedere flexibilitatea oferita de PyTorch in ceea ce priveste transformarile si antrenarea, intregul proiect a fost adaptat conform acestui framework. Initial arhitectura CNN a fost pastrata, dar pentru ca aveam foarte multe date cu transformari identice am decis sa profitam de functiile random asa ca am format un nou set de date compus din: normalizare; Resize + FiveCrop + selectie aleatorie din cele 5 crop-uri; RandomChoice pentru rotatii (0, 90, 180, 270) + ColorJitter; RandomCrop + flipuri orizontale si verticale. Modelul a depasit pentru prima data pragul de 93% acuratete.

Dat fiind ca nu mai aveam atatea date am redus complexitatea modelului arhitectural si am decis sa folosim Dropout-ul gradual pentru a avea pierderi ponderate, modelul devenind astfel: 3x64, 64x64, Dropout 0.25; 64x128, 128x128, Dropout 0.3; 128x256, 256x256, Dropout 0.4; 256x384, 384x384, Dropout 0.5. Intre toate straturile s-a pastrat BatchNormalization si MaxPooling, activarea fiind ReLU. Optimizatorul folosit a fost imbunatatit: AdamW, cu penalizare weight_decay = 1e-4 care este mai potrivita pentru CosineAnnealing deoarece o sa incercam sa mergem pana la o valoare minima de 1e-6. Aceaste imbunatatiri au condus modelul la acuratete de 93.68% pe validare si 93.533% pe competitie.



Augumentariile au avut de la inceput pana la sfarsit un rol important in special rotirea, luminozitate, contrastul si hue-ul, dar si crop-urile asa ca am adaugat o noua transformare care contine putin din fiecare : RandomRotation() + RandomAffine() + ColorJitter(). Acesta augumentare noua ne-a dus la 94.28% validare si 93.86% pe competitie.

Dupa ce am lasat modelul sa se antreneze 250 de epoci de 3 ori, am observat ca din cauza augumentarii cu metode random, rata de precizie este afectat de seed-ul pe care il primim asa ca am salvat intr-un folder toate modelele de peste 93.3%



Incercand sa mai modificam modelul arhitectural: am adugat 384x512, 512x512, dar a scazut probabilitate pana la maxim 92.90% pe validare asa ca am incercat sa modificam logica de selectie a label-urilor si am setat label_smoothing = 0.1. Aceasta imbunatatire ne-a adus stabilitate in a avea cat mai multe scoruri de peste 93.3%. Cel mai bun rezultat pe care l-am primit a fost 94.92% rata de validare si 93.93% in competitie.

In final aveam foarte multe modele cu acuratete peste rezultatul nostru de 93.93% asa ca am selectat 100 de modele intre 93.8% si 94.92% si le-am pus sa voteze si acelasi procedeu l-am aplicat si cu cele cuprinse intre 94.50% si 94.92%. Am obtinut in competitie 94.06% respectiv 94.00%.

La finalul competitie s-au adevarit a obtine 93.18%.

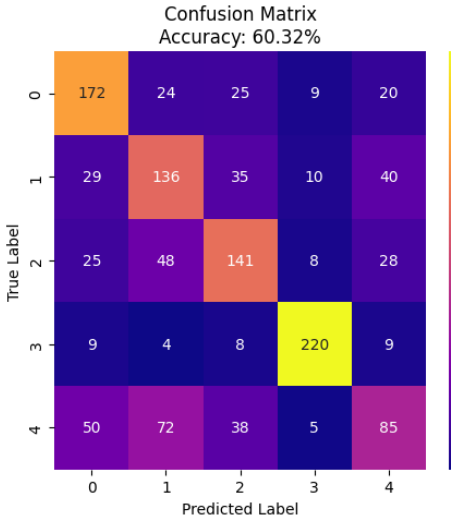
3. SVM

Am ales ca al doilea model sa fie Suport Vector Machine cu implementarea din libraria sklearn. SVM-urile au la baza functii kernel care transpun punctele din spatii neliniar separabile in spatii cu mai multe dimensiuni unde devin liniar separabile.

Primul pas este prelucrarea datelor si aplicarea unor filtre relevante pentru imaginile pe care le-am primit. Stim de la CNN ca luminozitate si contrastul au un rol important asa ca vom aplica aceste filtre imaginilor, iar apoi le vom normaliza:

Original **Luminozitate+Contrast** **Normalizare**

Al doilea pas este prelucrarea datelor intr-un vector unidimensional pentru scalare. Scalarea din libraria sklearn este o tehnica fundamentala de normalizare pe care am aplicat-o si la modelul CNN. Pentru ca fiecare imagine are 30000 de pixeli am aplicat Principal Component Analysys(PCA) pentru reduce dimensiunea. Putem observa urmatorea matrice de confuzie:

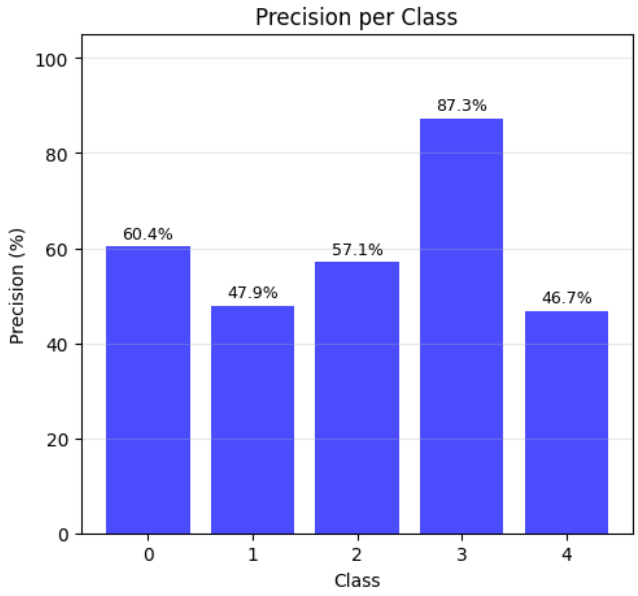


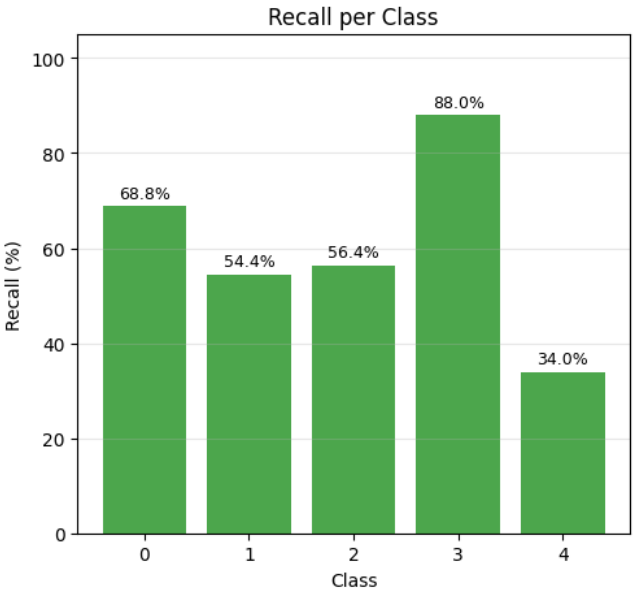
Kernel-ul Radial Basis Function(RBF) calculeaza distanta dintre doua puncte folosind distanta euclidiana, avand formula: $K(x, y) = e^{\left(-\gamma||x-y||^2\right)}$. Trebuie sa determinam atat paramentru pentru PCA, adica cate componente principale o sa avem, cat si C-ul care este paramentru responsabil de regularizare:

PCA Components	Explained Variance	C Parameter	Train Accuracy (%)	Validation Accuracy (%)	Overfitting (%)
200	**0.698**	0.001	41.51	40.56	0.95

200	0.698	0.01	42.87	42.00	0.87
200	0.698	0.1	57.94	54.00	3.94
200	0.698	**1.0**	81.32	**60.32**	21.00
200	0.698	10.0	99.75	59.60	40.15
200	0.698	100	100.00	58.72	41.28
200	0.698	1000	100.00	58.72	41.28
300	**0.746**	0.001	40.58	40.40	0.18
300	0.746	0.01	41.90	40.24	1.66
300	0.746	0.1	57.62	53.68	3.94
300	0.746	1.0	83.05	60.24	22.81
300	0.746	**10.0**	99.86	**60.32**	39.54
300	0.746	100	100.00	59.92	40.08
300	0.746	1000	100.00	59.92	40.08
400	**0.779**	0.001	40.26	40.00	0.26
400	0.779	0.01	41.18	39.92	1.26
400	0.779	0.1	57.46	52.80	4.66
400	0.779	**1.0**	84.28	**60.32**	23.96
400	0.779	10.0	99.92	59.52	40.40
400	0.779	100	100.00	59.76	40.24
400	0.779	1000	100.00	59.76	40.24
500	**0.803**	0.001	40.10	39.92	0.18
500	0.803	0.01	40.81	39.92	0.89
500	0.803	0.1	57.30	53.04	4.26
500	0.803	1.0	84.93	60.16	24.77
500	0.803	10.0	99.94	59.68	40.26
500	0.803	100	100.00	59.36	40.64
500	0.803	1000	100.00	59.36	40.64

Putem observa din acest tabel ca cel mai bun model are 60.32%, iar pe kaggle acesta a obtinut 61.4%, iar la finalul competitiei s-a adevarit a fi 59.56%.


Precision per Class
Precision (%)
100
80
60
40
20
0
60.4%
47.9%
57.1%
87.3%
46.7%
0
1
2
3
4
Class


Recall per Class
Recall (%)
100
80
60
40
20
0
68.8%
54.4%
56.4%
88.0%
34.0%
0
1
2
3
4
Class