Deepfake Image Classification

In primul rand m-am uitat era dimensiunea si structura datelor de train si validation, 12500 vs 1250. Dupa m-am uitat pe distributia celor 5 clase si am observant ca dataset e perfect echilibrat (si train si validation).

Train data:	Validation data:
0 2500	3 250
2 2500	0 250
3 2500	2 250
1 2500	1 250
4 2500	4 250

Dupa am definit functia show_class_images ca sa pot vizualiza imagini din fiecare clasa.

Inainte sa incerc un model SVM pentru clasificare, am preprocesat datele.

Am definit functia load_images care va citi fiecare imagine dintr-un folder, va transforma imaginea in vector ulterior folosind reshape(1, -1) pentru a transforma matricea 3d (width, height, color_channels) in vector 1d (flatten). Impreuna cu imagini preprocesate am returnat si array de labels (corespunzator id-ului imaginii).

Am normalizat imagini preprocesate cu StandardScaler.

Dupa am folosit svm.SVC cu kernel rbf si gamma auto. Modelul acesta a obtinut acuratetea 0.6336.

Asa arata classification report pentru svm:

Class	Precision	Recall	F1-Score	Support
0	0.664179	0.712000	0.687259	250

Class	Precision	Recall	F1-Score	Support
1	0.506073	0.500000	0.503018	250
2	0.637993	0.712000	0.672968	250
3	0.915612	0.868000	0.891170	250
4	0.429224	0.376000	0.400853	250
Accuracy			**0.6336**	
Macro Avg	0.630616	0.633600	0.631054	1250
Weighted Avg	0.630616	0.633600	0.631054	1250

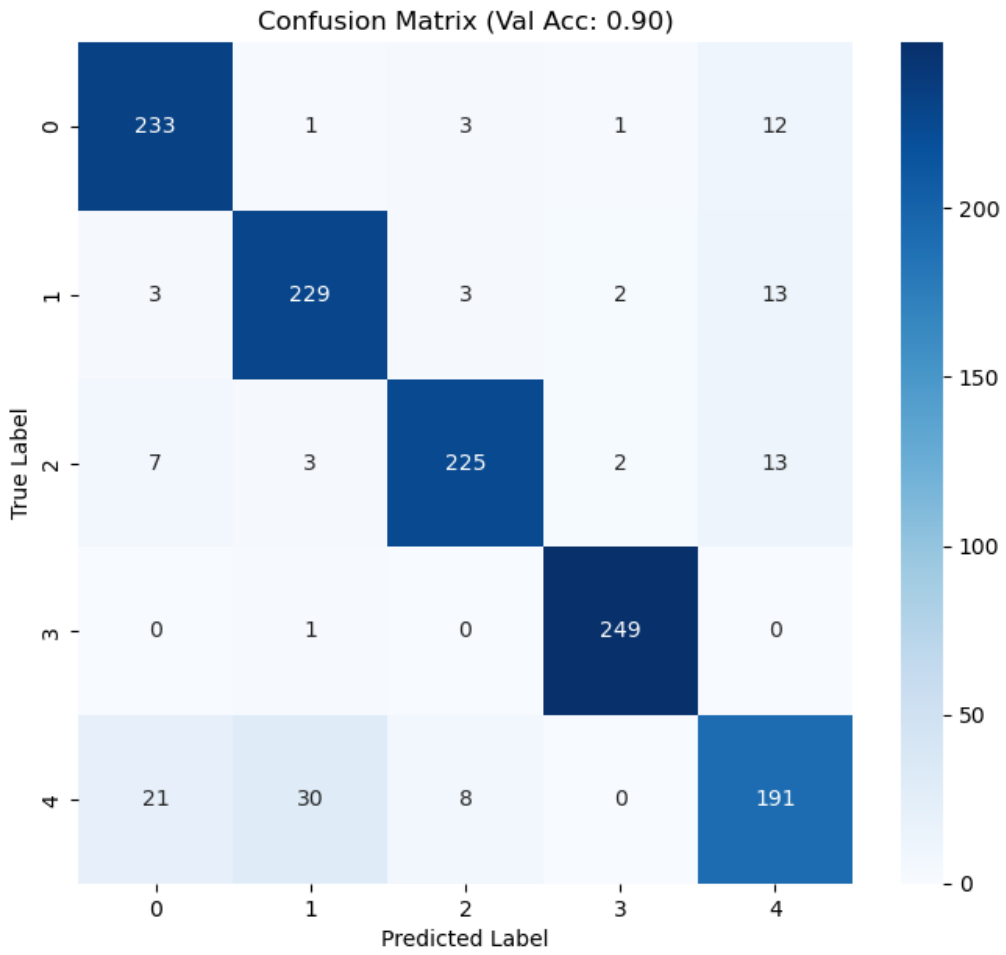
Dupa ce am privit mai multe tutoriale, am inteles ca fara utilizare de CNN, rezultatele nu vor fi foarte bune. Deci am decis sa fac un CNN model, pe care sa-l antrenez. Primul CNN pe care am inceput sa-l antrenez e DeepFakeNet. El are 3 blocuri care cuprind convolutii, max pooling si functii de activare non liniare (ReLU sunt cele mai populare si folosite acum). Numarul de canaluri pentru conv2d creste astfel: 3(rgb input) -> 32 -> 32 -> 64 -> 64 -> 128 -> 128. In fiecare bloc se aplica max pool 2d care reduce dimensiunile imaginii cu factor de 2. (128,128) -> (64, 64) -> (32, 32) -> (16, 16). Convolutii cu padding=1 pastreaza dimensiunea imaginii dupa transformare. In sfarsit folosesc global adaptive pool care face transformare (B, 128, 16, 16) -> (B, 128, 1, 1). Stratul final e linear care primeste ca input (B,128) de features din layer precedent si produce ca output 5 valori, utilizate pentru clasificare. La fiecare bloc se utilizeaza batch normalization pentru stabilizare si dropout layers pentru a evita overfit.

Am utilizat custom dataset care va transforma imagini de train (data augmentation) folosind diferite transformari, precum RandomHorizontalFlip, RandomRotation, ColorJitter etc. Imaginile sunt convertite in tensor de dimensiune

(128, 128). Am folosit SGD optimizer si cross entropy loss cu valori default. Pana la urma am obtinut acuratetea 0.9016 dupa mai multe epoci.

Class	Precision	Recall	F1-Score	Support
0	0.882576	0.932000	0.906615	250
1	0.867424	0.916000	0.891051	250
2	0.941423	0.900000	0.920245	250
3	0.980315	0.996000	0.988095	250
4	0.834061	0.764000	0.797495	250
Accuracy	**0.9016**			
Macro Avg	0.901160	0.901600	0.900700	1250

Confusion matrix:



Am observat ca clasa 4 are scoruri foarte mici, recall fiind cel mai mic. Am decis in modelul meu sa schimb dropout rates la stratul 2 si 3. Am schimbat optimizer pe Adam (in diferite surse s-a demonstrate ca el e mai efficient decat SGD), am scazut learning rate, i-am clasei 4 un weight mai mare. Am adaugat scheduler pentru gestiunea ratei de invatare mai buna. Am adaugat Gaussian Blur transform si am adaugat normalizarea imaginilor cu valori intre -1 si 1 (normal distribution).

```
transforms.Normalize(mean=[0.5, 0.5, 0.5],
                     std=[0.5, 0.5, 0.5])
```

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)
```

weights = torch.tensor([1.0, 1.0, 1.0, 1.0, 2.7])

weights = weights.to(device)

criterion = nn.CrossEntropyLoss(weight=weights)

scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='max', patience=2, factor=0.5)

Pana la urma am ajuns la scorul pe validare in jurul lui 0.92

100%|  | 391/391 [01:21<00:00, 4.82it/s]

Epoch 1: Loss=0.2987, Val Acc=0.9200

100%|  | 391/391 [01:19<00:00, 4.89it/s]

Epoch 2: Loss=0.2927, Val Acc=0.9232

100%|  | 391/391 [01:25<00:00, 4.56it/s]

Epoch 3: Loss=0.2933, Val Acc=0.9216

100%|  | 391/391 [01:22<00:00, 4.73it/s]

Epoch 4: Loss=0.3011, Val Acc=0.9208

Am observant ca accuracy si loss nu mai scad, deci e un semn de overfit. Am decis sa trec pe alt model CNN, VGG16, un model popular. L-am antrenat cu Adam optimizer cu scheduler din modelul precedent si cross entropy loss. Pe parcursul

epocilor scadeam learning rate manual si adjustam weights vector in functie de classification report precision/recall scores la diferite clase. Am adaugat si CutMix or MixUp data augmentation. La un moment am incercat sa folosesc si weighted sampler pentru class 4, dar rezultatul a fost prost (accuracy < 90%)

Am obtinut tot acuratetea in jurul lui 0.92:

Class	Precision	Recall	F1-Score	Support
0	0.913386	0.928000	0.920635	250
1	0.926829	0.912000	0.919355	250
2	0.928571	0.936000	0.932271	250
3	0.995968	0.988000	0.991968	250
4	0.840000	0.840000	0.840000	250
Accuracy			**0.9208**	
Macro Avg	0.920951	0.920800	0.920846	1250
Weighted Avg	0.920951	0.920800	0.920846	1250

Acum am decis ca pot sa obtin un scor mai bun combinand aceste 2 modele: DeepFakeNet2 si VGG16 (ensemble model). Am construit modelul urmand tutorialul: https://medium.com/@alexppppp/how-to-train-an-ensemble-of-convolutional-neural-networks-for-image-classification-8fc69b087d3

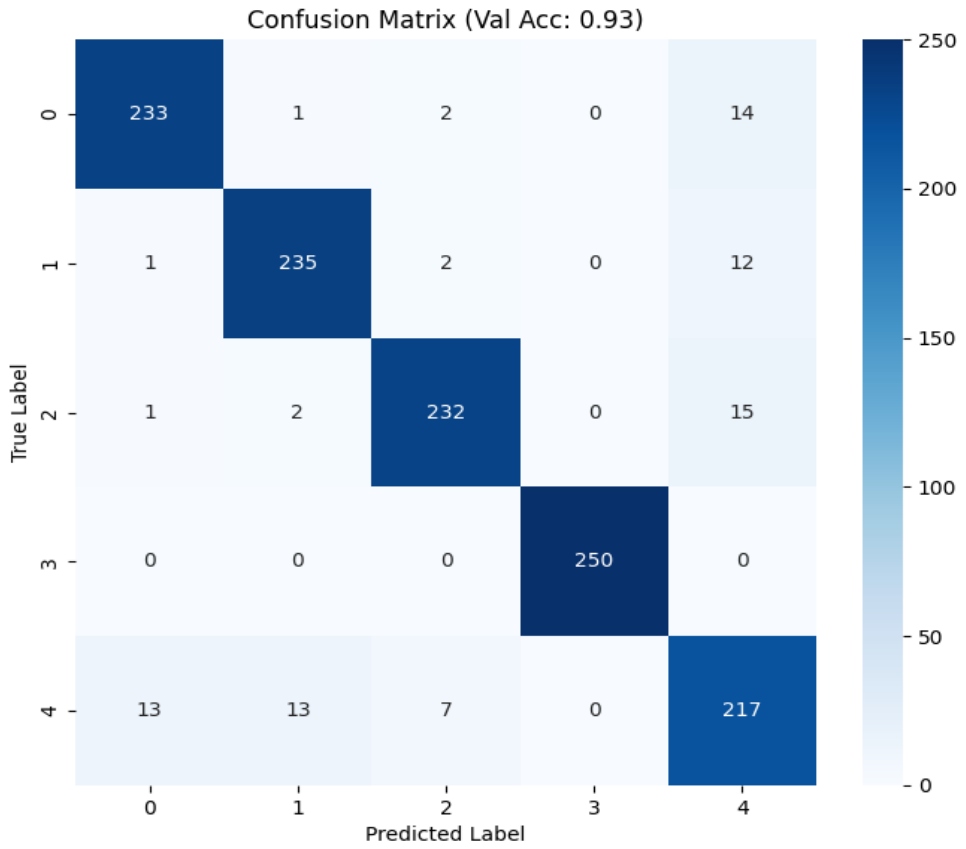
Antrenand model ansamblu, eu am ajuns la acuratetea pe validare 0.9336 si 0.9413 pe test submission. Asa arata raportul de clasificare a modelului obtinut:

Class	Precision	Recall	F1-Score	Support
0	0.939516	0.932000	0.935743	250
1	0.936255	0.940000	0.938124	250
2	0.954733	0.928000	0.941176	250
3	1.000000	1.000000	1.000000	250
4	0.841085	0.868000	0.854331	250

Class	Precision	Recall	F1-Score	Support
Accuracy			**0.9336**	
Macro Avg	0.934318	0.933600	0.933875	1250
Weighted Avg	0.934318	0.933600	0.933875	1250

Am dat de data aceasta ponderea mai mare claselor 0 si 4 (1.9 si 1.7), caci in process de antrenare aceste clase aveau un recall mai mic decat precision.

Confusion matrix pentru model ansamblu:



Dupa cum se observa, sumele valorilor pe ultima linie si ultima coloana au devenit mai balansate, ce inseamna ca s-a stabilit un echilibru mai bun intre precision si recall pentru clasa 4.

Concluzie: ansamblu de modele poate da rezultate mai bune decat fiecare model aparte.

Concluzie globala

Dupa utilizarea si testarea mai multor modele, se observa ca retele neuronale convolutionale sunt mai eficiente in clasificarea imaginilor decat modele traditionale precum SVM. Tehnicile precum augmentarea datelor, gestionarea ratei de invatare, dropout si utilizarea ponderilor pentru clase contribuie la rezultatele mai bune ale modelului. Un ansamblu de 2 sau mai multe modele e de multe ori mai eficient decat fiecare model aparte. Probabil ar fi bine de adaugat mai multe straturi in retelele neuronale testate sau de facut un ansamblu de mai multe modele, insa asta ar insemna cresterea complexitatii computationale.