Documentatie

Dumitrescu Delia Ioana - grupa 234

Random Forests

Un clasificator folosit a fost *RandomForestClassifier* din *sklearn.* Acesta foloseste arbori de decizie. Se creeaza un numar fix de arbori de decizie tinand cont de impuritatea unui split(cat de “bine” separa clasele pe baza unor features). Mai apoi, fiecare arbore “voteaza” in ce clasa se incadreaza o anumita poza. In functie de votul majoritar, se decide clasa prezisa.

Pentru prima incercare, au fost folositi 100 de arbori de decizie.

Functia care calculeaza cat de bun este un split a fost *gini,* care este egala cu:

$$\text{Gini} = 1 - \sum_{i=1}^{n} (p_i)^2$$

Numarul de features care au fost luate in considerare pentru crearea unui split este:

```
max_features=sqrt(n_features)
```

Astfel, s-a obtinut o acuratete de 0.6417777777777778.

Pentru a doua incercare, initial au fost pastrati aceeasi parametrii, insa imaginile au fost preprocesate, fiind standardizate. Aceasta standardizare a adus o imbunatatire, insa destul de mica, ajungandu-se la o acuratete de 0.6455555555555555.

Mai departe, am marit numarul de arbori de decizie folositi de la 100 la 500. Aceasta schimbare a imbunatatit acuratetea cu peste 1%, ajungand la: 0.666.

Alta incercare a fost schimbare functiei care determina impuritatea unui arbore din *gini* in *entropy,* a carei formula este:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

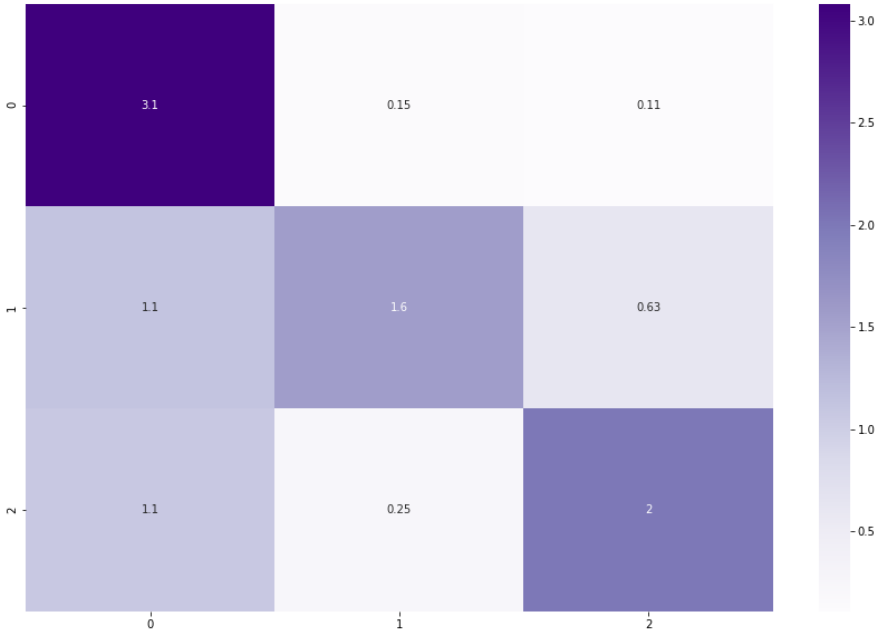
Aceasta schimbare a scazut acuratetea pana la 0.6453333333333333, asa ca am renuntat la ea si am pastrat *gini.* Am incercat, de asemenea, sa luam in considerare

```
max_features=log2(n_features)
```

insa acuratetea a scazut la 0.6042222222222222.

Prin urmare, rezultatele obtinute folosind *Random Forests* cu *500 de arbori de decizie*, functia *gini* si *sqrt(features)* considerate pentru cautarea unui split au fost:

Validation Accuracy: 0.666
Confusion Matrix:

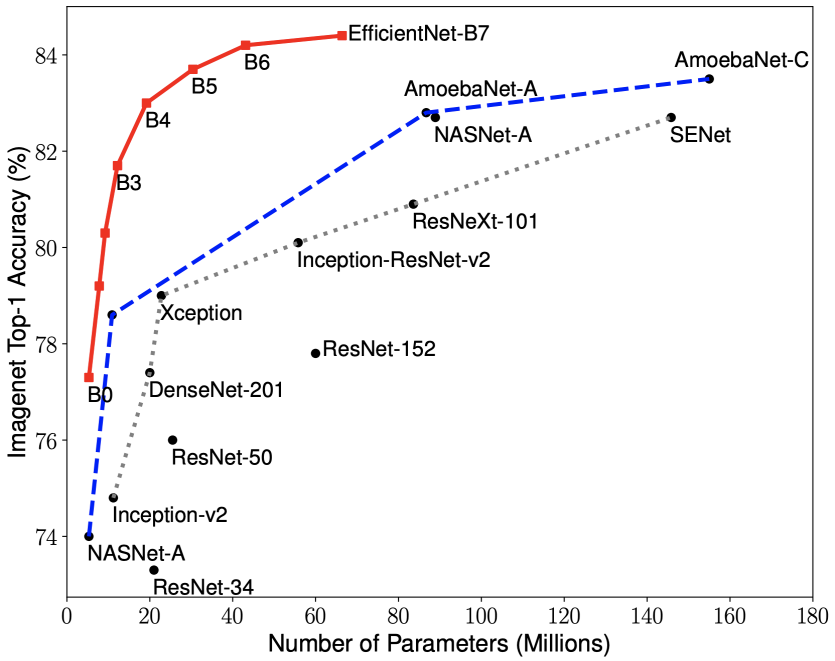


Transfer Learning

A doua abordare a fost folosirea unei retele neuronale convolutionale. Alegerea arhitecturii nu este, totusi, una triviala, asa ca am decis sa caut o retea neuronala deja construita si preantrenata, iar mai apoi sa folosesc *Transfer Learning* pentru a antrena reteaua pe datele pe care le avem la dispozitie.

Reteaua aleasa a fost *EfficientNet-B0. EfficientNet* este o familie de retele *(B0-B7)* dezvoltata de *Google*, iar cea mai performanta a avut in 2020 o acuratete de 88.61% pe *ImageNet*, un set de peste 14 milioane de poze adnotate.

Arhitectura lui *EfficientNet* nu a fost aleasa de oameni, ci de o retea. Ei au dezvoltat acest model folosind o cautare ce avea ca obiectiv atat imbunatatirea acuratetii, cat si optimizarea arhitecturii in sine.

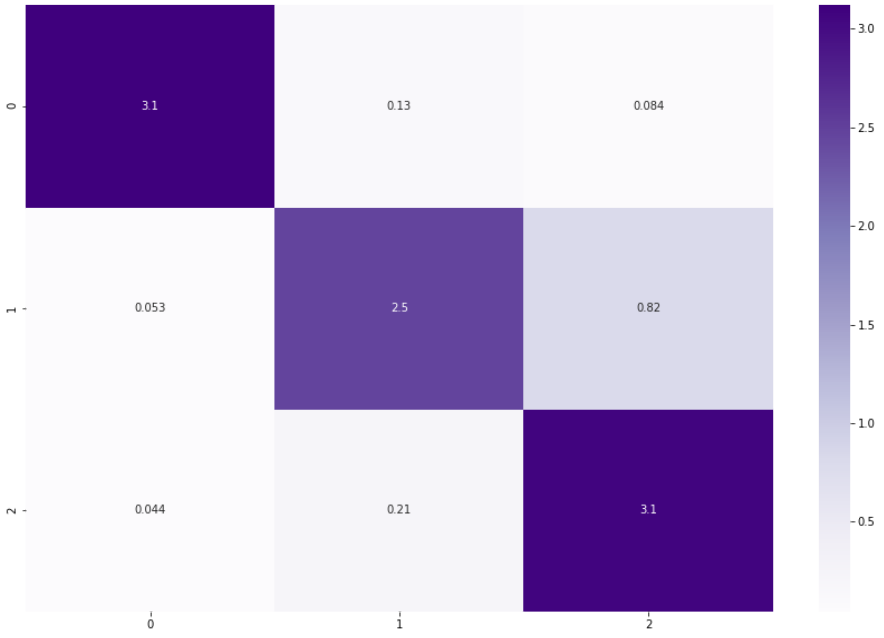


Preprocesarea imaginilor a constat in transformarea imaginilor in *PILImage,* apoi un *resize* la *(224, 224)* si transformarea imaginilor in *tensor.* Datele au fost incarcate in batch-uri de cate 64 de imagini fiecare, care au fost shuffle-uite la fiecare epoca. Pentru calcularea *loss-ului,* am folosit *CrossEntropyLoss*, iar pentru *optimizer* am ales *Adam* cu un *learning rate* de 1e-5. Am antrenat timp de 20 de epoci si am calculat acuratetea pe validare, obtinand: 0.8306666666666667.

Loss-ul scadea foarte incet, asa ca am decis sa maresc *learning rate-ul*. Am schimbat marimea unui batch la 32 si am modificat *learning rate-ul* la 1e-4. Dupa 3 epoci, modelul a inceput sa faca overfit, asa ca l-am pastrat pe cel de la epoca 3 si am calculat acuratetea pe validare, obtinand 0.8644444444444445.

Prin urmare, rezultatele obtinute folosind o *retea preantrenata* cu un *learning rate* de 1e-4, *CrossEntropyLoss* pentru calcularea *loss-ului* si *Adam* ca *optimizer* sunt:

Validation Accuracy: 0.8644444444444445
Confusion Matrix:


3.1
0.13
0.084
0.053
2.5
0.82
0.044
0.21
3.1
0
1
2
3.0
2.5
2.0
1.5
1.0
0.5