Clasificare de imagini in 5 clase

1. Introducere

Acest proiect isi propune sa realizeze clasificarea unor imagini color in 5 clase folosind metode de invatare automata. Setul de date contine imagini RGB, toate cu dimensiunea de 100x100 pixeli. Fisierele CSV contin identificatorii imaginilor si, in cazul seturilor de antrenament si validare, etichetele corespunzatoare.

2. Primul model: Multi-Layer Perceptron

- Am scris propria clasa pentru ImageDataset (care extinde clasa Dataset) pentru ca am avut nevoie sa rescriu metodele \_\_len\_\_ si \_\_getitem\_\_.

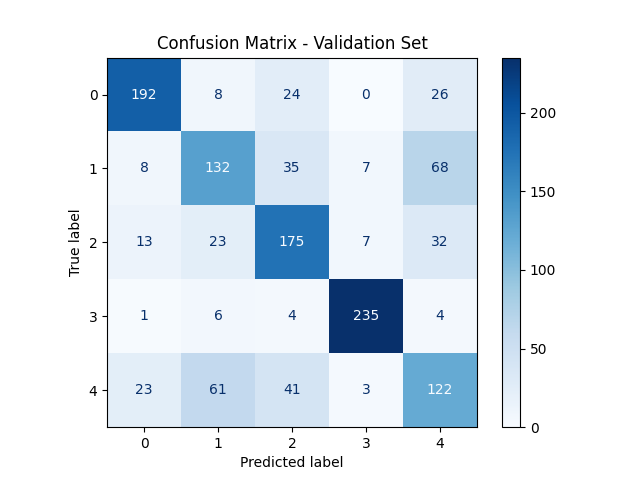
- Imaginile au fost incarcate ca tensori RGB (3 canale) si transformate in vectori unidimensionali prin turtire.

- Arhitectura modelului:

* Flatten
* Primul strat: Linear(30000 → 512) + ReLU
* Al doilea strat: Linear(512 → 512) + ReLU
* Ultimul strat: Linear(512 → 5)
* Functia de pierdere: CrossEntropyLoss
* Optimizator: SGD
* Learning rate: 1e-2
* Numar epoci:10 epoci
* Batch size: 64
* Nu am aplicat tehnici de augmentare.
* Nu am folosit alte reprezentari ale caracteristicilor.

In urmatorul tabel sunt descrise incercarile pe care le-am rulat pentru a gasi optimizer-ul, learning rate-ul, batch size-ul cele mai potrivite, pornind de la modelul actual, prima linie din tabel.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Learning rate | Optimizer | No. of epochs | Batch size | Accuracy |
| 1e-2 | SGD | 10 | 64 | 52 |
| 1e-3 | SGD | 10 | 64 | 44 |
| 1e-1 | SGD | 10 | 64 | 37 |
| 1e-3 | SGD | 30 | 64 | 53 |
| 1e-2 | SGD | 30 | 64 | 52 |
| 1e-2 | Adam | 10 | 64 | 20 |
| 1e-3 | Adam | 10 | 64 | 56 |
| 1e-4 | Adam | 10 | 64 | 51 |
| 1e-5 | Adam | 10 | 64 | 50 |
| 1e-3 | AdamW | 10 | 64 | 54 |
| 1e-3 | Adam | 20 | 64 | 53 |
| 1e-3 | Adam | 30 | 64 | 56 |
| 1e-3 | Adam | 10 | 32 | 50 |
| 1e-3 | Adam | 10 | 128 | 55 |
| 1e-3 | Adam | 20 | 128 | 54 |
| 1e-3 | Adam | 30 | 128 | 54 |

Dupa aceste incercari, am ramas cu rata de invatare 1e-3, Adam, 10 epoci, si 128 batch size, si am decis sa augmentez modelul. Am adaugat o augmentare simpla cu random horizontal flip, dropout intre straturi ca sa nu suprainvete pe datele limitate, batch normalization dupa straturile liniare ca sa stabilizez si sa accelerez antrenamentul, scheduler care reduce rata de invatare dupa cateva epoci daca pierderea nu scade, si in final am adaugat un al treilea strat intermediar de neuroni si am modificat numarul lor, si am ajuns la 58 acuratete. Am adaugat random rotation si color jitter dar a coborat la 55, asa ca le-am eliminat, ramanand doar cu random horizontal flip, si 58 acuratete. Apoi am adaugat random resized crop si am ajung la 60. Parametrii pentru RandomHorizontalFlip, RandomResizedCrop, Dropout si ReduceLROnPlateau au fost testati separat, fiecare cu valori usor mai mici si mai mari decat cele pastrate in versiunea finala. Niciuna dintre variantele alternative nu a produs rezultate mai bune pe setul de validare, asa ca am pastrat configuratia care a dat cea mai buna acuratete generala. Am incercat si weight\_decay dar nu parea sa aduca niciun bonus, asa ca am renuntat. In final, am introdus early stopping, cu 50 epochs si 5 patience, si am depasit pragul autoimpus de 70% in competitie, asa ca am trecut la urmatorul model.

Modelul final

- Rezultate:

* Acuratete pe validation: 69.20%
* Pierdere: 0.777047
* Matrice de confuzie:

- Arhitectura modelului:

* Flatten
* Primul strat: Linear(30000, 1024) + ReLU
* BatchNorm1d(1024)
* Dropout(p=0.4)
* Al doilea strat: Linear(1024, 1024) + ReLU
* BatchNorm1d(1024)
* Dropout(p=0.4)
* Al treilea strat: Linear(1024, 512) + ReLU
* BatchNorm1d(512)
* Dropout(p=0.4)
* Ultimul strat: Linear(512, 5)
* Functia de pierdere: CrossEntropyLoss
* Optimizator: Adam
* Learning rate: 1e-3
* Numar epoci: 50 epoci
* Early Stop Patience: 5
* Batch size: 128
* Augmentare: RandomResizedCrop(100, scale=(0.8, 1.2)) si RandomHorizontalFlip(p=0.5)
* Scheduler: ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=3)

3. Al doilea model: Convolutional Neural Network

- Aproape identic cu primul model: am schimbat doar clasa NeuralNetwork, inlocuind toate straturile Linear (in afara de ultimul strat ascuns si cel de output) cu straturi Conv2d, mutand Flatten inainte de stratul de output. In plus, am mai adaugat un strat ascuns de Conv2d

- Toate celelalte augmentari, optimizari, etc. sunt pastrate din ultima varianta a modelului MLP.

- Arhitectura modelului:

* Primul strat: Conv2d(3, 32, kernel\_size=3, padding=1) + ReLU
* BatchNorm2d(32)
* MaxPool2d(2)
* Dropout(p=0.4)
* Al doilea strat: Conv2d(32, 64, kernel\_size=3, padding=1) + ReLU
* BatchNorm2d(64)
* MaxPool2d(2)
* Dropout(p=0.4)
* Al doilea strat: Conv2d(64, 128, kernel\_size=3, padding=1) + ReLU
* BatchNorm2d(128)
* MaxPool2d(2)
* Dropout(p=0.4)
* Al treilea strat: Conv2d(128, 256, kernel\_size=3, padding=1) + ReLU
* BatchNorm2d(256)
* AdaptiveAvgPool2d((4, 4))
* Dropout(p=0.4)
* Al patrulea strat: Linear(256 \* 4 \* 4, 512) + ReLU
* BatchNorm1d(512)
* Dropout(p=0.4)
* Flatten
* Ultimul strat: Linear(512, 5)
* Functia de pierdere: CrossEntropyLoss
* Optimizator: Adam
* Learning rate: 1e-3
* Numar epoci: 50 epoci
* Batch size: 128
* Augmentare: RandomResizedCrop(100, scale=(0.8, 1.2)) si RandomHorizontalFlip(p=0.5)
* Scheduler: ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=3)

Rezultatele au aratat o imbunatatire mare fata de MLP, acuratete de 82%. Pentru acest model tinta autoimpusa era de 90% acuratete. Fiind deja aproape de ea, si hiperparametrii fiind deja tunati de la modelul anterior, am incercat sa modific valori din clasa NeuralNetwork, incepand cu Dropout. In primul rand am setat Dropout mai drastic pentru ultimul strat, de 0.5, si in rest mai lejer, de 0.3, si am ajuns la 84. Apoi nu am mai schimbat Dropout-ul pentru ultimul strat, ci doar in rest. Am incercat 0.2 si am ajuns la 87, apoi cu 0.1 si am ajuns la 89. Am dat submit si am ajuns la 91% in competitie. Nu am mai testat alti parametri, in afara de weight\_decay la optimizer. Cu 1e-4 nu a modificat mult, iar loss-ul parea putintel mai bun, dar cu 1e-3 si 1e-5 a mers mai prost ca inainte, asa ca l-am lasat cu 1e-4. Cele 2 submisii selectate in competitie pentru evaluarea finala sunt CNN fara weight\_decay si CNN cu weight\_decay A diagram of a number and a number

AI-generated content may be incorrect.(varianta pe care o sa o descriu la sectiunea Modelul Final).

Modelul final

- Rezultate:

* Acuratete pe validation: 89.44%
* Pierdere: 0.271512
* Matricea de confuzie:

- Arhitectura modelului:

* Primul strat: Conv2d(3, 32, kernel\_size=3, padding=1) + ReLU
* BatchNorm2d(32)
* MaxPool2d(2)
* Dropout(p=0.1)
* Al doilea strat: Conv2d(32, 64, kernel\_size=3, padding=1) + ReLU
* BatchNorm2d(64)
* MaxPool2d(2)
* Dropout(p=0.1)
* Al doilea strat: Conv2d(64, 128, kernel\_size=3, padding=1) + ReLU
* BatchNorm2d(128)
* MaxPool2d(2)
* Dropout(p=0.1)
* Al treilea strat: Conv2d(128, 256, kernel\_size=3, padding=1) + ReLU
* BatchNorm2d(256)
* AdaptiveAvgPool2d((4, 4))
* Dropout(p=0.1)
* Al patrulea strat: Linear(256 \* 4 \* 4, 512) + ReLU
* BatchNorm1d(512)
* Dropout(p=0.5)
* Flatten
* Ultimul strat: Linear(512, 5)
* Functia de pierdere: CrossEntropyLoss
* Optimizator: Adam
* Learning rate: 1e-3
* Weight decay: 1e-4
* Numar epoci: 50 epoci
* Batch size: 128
* Augmentare: RandomResizedCrop(100, scale=(0.8, 1.2)) si RandomHorizontalFlip(p=0.5)
* Scheduler: ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=3)