INTELIGENTA ARTIFICIALA

Clasificarea imaginilor cu Rețele Neuronale Convolutionale: De la MLP la CNN în competiția Kaggle

Student : Andruta Andra Mihaela 232

Universitatea din București

Facultatea de Matematică și Informatică

2025

CUPRINS

DESCRIEREA ABORDĂRII

Procesul meu de rezolvare a taskului propus în cadrul competiției a început prin revizuirea aprofundata a notiunilor invatate în cursuri, seminare și laboratoare.Fiind o competiție care a venit la scurt timp după dobândirea acestor cunoștințe, am considerat esențial să plec de la bazele solide oferite de suportul educațional al facultății.

Fiind la început de drum în domeniul machine learning și, în special, al rețelelor neuronale, am considerat esențial sa ma reintorc la bazele solide ce mi-au fost oferite ca suport educațional la facultate.

În aceasta etapa, am reluat exemplele din laboratoare, am revizuit codul exercițiilor rezolvate de mine la acel moment și am încercat sa aprofundez conceptele cheie, atât din suporturile oferite ,dar și surse suplimentare precum articole online și explicații suplimentare oferite de OpenAI, mai ales pentru ca eu invatam aceste noțiuni înaintea predarii lor efective la laborator, tocmai din dorinta de învățare și rezolvare a taskului din competiție treptat, nu presata de timp.

Astfel, după o perioada de revizuire , prima versiune a modelului meu a fost un Multi-Layer Perceptron(MLP) inspirat direct din laboratorul 6, pe care am incercat sa il imbunatatesc treptat, prin abordari pe care eu le consideram relevante și care porneau tocmai din informațiile pe care le-am dobândit pe parcursul semestrului.

Am construit, am testat, apoi am construit din nou și din nou tot felul de arhitecturi, pastrandu-mi fiecare schimbare într-un fișier separat,relevant pentru schimbări, comparand de fiecare data performanțele pe care orice schimbare mi-o aducea,fie ca performanța era într-un mod pozitiv sau negativ.

În toate încercările mele am folosit datele oferite de competite,fără pre trained sau librării externe. Am aplicat argumentări ale imaginilor, am experimentat modificări de structura (dimensiunea layerelor, funcții de activare, dropout, etc), însă în ciuda tuturor acestor ajustări, MLP-ul meu ramanea blocat la o performanta de maxim 62%, ceea ce aducea de la sine o clasare joasă. În paralel, am observat ca alți colegi atingeau scoruri semnificativ mai mari , iar acest context m-a motivat sa ies din zona de confort, din zona de siguranța oferită de laborator și să încep sa mă documentez serios pe tema rețelelor neuronale convolutionale (CNN), pentru ca fiind vorba de o competiție dacă vreau sa încerc sa mă ridic la nivelul celorlalți care probabil au mers pe documentare externă de la bun început,trebuie sa îi urmez, nu mă pot limita, pentru ca tocmai acesta este de fapt scopul academic al universității sa ne autodepasim,și sa reusim sa asimilam cat mai multe informații de valoare pe care putem vedea dacă le-am înțeles doar dacă le punem în aplicare și ce context ar fi mai bun decat o competiție ca cea de pe kaggle.

Astfel am invatat conceptele de bază, cum arată o astfel de arhitectura din diverse articole sau site-uri din domeniu, și mai apoi am construit un prim CNN de la 0, cu o arhitectură inspirată din ceea ce știam despre extragerea automată de caracteristici prin straturi convoluționale. La fel ca în cazul MLP-ului, am testat multiple variante, păstrând o evidență clară a fiecărei modificări.

Rezultatul a fost o îmbunătățire substanțială, scorul meu crescând cu aproximativ 25%, ducandu-ma în zona de 86% după încă câteva mici schimbări,ceea ce a reprezentat o confirmare importantă ca direcția aleasă , schimbarea făcută este una corecta. Modelul a fost antrenat strict pe datele din competiție, fără utilizarea unor modele pre antrenate sau seturi de date externe, în conformitate cu regulile impuse.

Această tranziție de la un MLP simplu la un CNN a fost un proces esențial de învățare și dezvoltare personală. Competiția nu a fost doar un exercițiu tehnic, ci și o provocare reală de autodepășire, care m-a forțat să învăț rapid, să experimentez curajos și să documentez riguros fiecare pas făcut.

PREPROCESARE ȘI AUGMENTARE

Așa cum am menționat și în descrierea abordării, în toate etapele proiectului am considerat esențială aplicarea unor tehnici de preprocesare și augmentare a imaginilor, indiferent de modelul utilizat. Acest lucru a fost inspirat de exemplele parcurse în laboratoare, unde normalizarea sau standardizarea imaginilor era o practică frecventă.

Totuși, această componentă nu a fost implementată într-o formă finală de la început, ci a evoluat odată cu înțelegerea mea mai bună a datelor și a modului în care preprocesarea afectează performanța modelelor.

Inițial, am ales să convertesc imaginile RGB în imagini grayscale (un singur canal), deoarece în laborator lucrasem exclusiv cu acest tip de date. În această primă versiune, normalizarea era aplicată pe canalul unic, cu medie 0.5 și deviație standard 0.5. Totuși, performanța obținută era modestă și mi-am dat seama că informația de culoare, prezentă în cele 3 canale RGB, ar putea fi importantă pentru clasificarea imaginilor.

După o analiză mai atentă a datelor și o serie de experimente comparative, am decis să păstrez cele trei canale de culoare. Această decizie s-a dovedit corectă, întrucât scorul pe setul de validare a crescut cu câteva procente.

Ulterior, am introdus și redimensionarea imaginilor la dimensiunea fixă de 128x128 pixeli.

În cele din urmă, după o documentare mai aprofundată, am adăugat și augmentari de imagine pentru a reduce overfitting-ul și a crește diversitatea datelor în timpul antrenării. Cele mai eficiente s-au dovedit a fi:

- **Flip orizontal aleatoriu** cu o probabilitate de 50% (RandomHorizontalFlip(p=0.5)),
- **Rotirea imaginilor** cu un unghi aleatoriu între -15 și 15 grade (RandomRotation(degrees=15)).

Transformările au fost implementate astfel, folosind biblioteca torchvision.transforms:[1]

```
img_preprocesare = transforms.Compose([
    transforms.Resize((128, 128)),
    # imaginea este oglindita stanga/dreapta cu o probabilitate de 50%
    transforms.RandomHorizontalFlip(p=0.5),
    # imaginea este rotită cu un unghi în intervalul [-15,15] grade
    transforms.RandomRotation(degrees=15),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5,0.5,0.5], std=[0.5,0.5,0.5])  # imaginea e normalizată între [-1, 1]
pe toate cele 3 canale RGB
])
```

Această combinație de tehnici de preprocesare și augmentare a contribuit semnificativ la îmbunătățirea performanței finale a modelului, asigurând o generalizare mai bună și crescută pe datele de validare și test.

De asemenea, am analizat posibilitatea utilizării unor metode de extragere manuală a trăsăturilor (feature engineering), precum histogramele de culoare sau descriitori de tip HOG/SIFT. Totuși, am ales să nu le includ în această etapă, întrucât scopul meu a fost să testez capacitatea modelelor de tip MLP și CNN de a învăța direct din imaginea brută. Având în vedere natura vizuală a datelor și puterea rețelelor neuronale de a extrage reprezentări abstracte, am considerat că utilizarea datelor RGB normalizate este suficientă și mai potrivită pentru o abordare end-to-end, fără intervenție manuală asupra trăsăturilor.

[1] PyTorch Vision Transforms – torchvision.transforms. Disponibil la: https://docs.pytorch.org/vision/stable/transforms.html

REPREZENTAREA CARACTERISTICILOR

Reprezentarea caracteristicilor a variat în funcție de modelul ales pentru clasificare.Deoarece am testat atât rețele neuronale complet conectate (MLP), cat și retele convolutionale (CNN), modul în care am tratat imaginile ca date de intrare a fost adaptat corespunzător.

1. MLP (Multi-Layer Perceptron)

Pentru modelul MLP, nu am extras caracteristici manual, ci am aplicat o strategie simplă de flattening a întregii imagini. Astfel, fiecare imagine RGB de dimensiune 128x128 a fost transformată într-un vector unidimensional de dimensiune 3 * 128 * 128 = 49152, care a fost apoi folosit ca input pentru rețea.

Această abordare este clasică pentru MLP-uri, unde toate valorile de intensitate ale pixelilor sunt tratate ca trăsături brute („raw features”), fără a ține cont de relațiile spațiale dintre ele.

```
self.input_flattener = nn.Flatten()
```

Acest tip de reprezentare funcționează, dar are limitări evidente în contextul imaginilor, deoarece ignoră complet structura locală și vecinătățile pixelilor. Performanța modelului s-a plafonat în jurul valorii de 62-64%.

2. CNN (Convolutional Neural Network)

În cazul modelului CNN, nu am extras explicit caracteristici, ci am lăsat rețeaua să le învețe automat prin intermediul straturilor convoluționale. Aceasta este una dintre principalele forțe ale CNN-urilor — capacitatea lor de a extrage automat caracteristici spațiale și semantice relevante din imagini, pe baza unor filtre antrenabile.

Astfel, inputul rețelei CNN a fost imaginea RGB procesată și augmentată, sub forma unui tensor de dimensiune [3, 128, 128]. Primele straturi convoluționale au învățat caracteristici de jos nivel (precum muchii, colțuri), iar straturile mai adânci au învățat trăsături mai abstracte, utile pentru clasificare.

```
CNN_AAM_X = self.pool(self.primul_layerCONV(CNN_AAM_X)) # 16x64x64
```

Nu am folosit niciun extractor clasic de trăsături (cum ar fi SIFT, HOG sau histograme de culoare), ci am lucrat exclusiv cu reprezentarea brută a imaginilor, bazându-mă pe puterea CNN-ului de a învăța aceste caracteristici direct din date.

MODELELE TESTATE

ARHITECTURA 1 : MLP (Multi-Layer Perceptron)

Un MLP este o rețea neuronală feedforward formată din cel puțin trei layere: un strat de input, unul sau mai multe straturi ascunse și un strat de output. Toți neuronii sunt complet conectați între straturi și funcția sa principală este de a învăța o mapare de tip input → output prin propagarea în față și înapoi.

Ca prim pas în abordarea acestei competiții, am ales să construiesc un model de tip Multi-Layer Perceptron (MLP), inspirat direct din laboratorul 6, cu scopul de a pune în aplicare noțiunile fundamentale de rețele neuronale învățate în cadrul cursului.

Scopul pe care l-am urmat în utilizarea unui MLP, a fost cel de a evalua dacă o rețea neuronala complet conectata poate face fata sarcinii de clasificare a imaginilor și am tratat aceasta abordare ca un baseline pentru comparatiile ce aveau să apară.

ARHITECTURA FINALĂ:

Modelul MLP final implementat a urmat o structură relativ simpla ,fiind compus din 3 straturi astfel:

1. Primul strat → 1024 neuroni
2. Al doilea strat → 512 neuroni
3. Al treilea strat → 256 neuroni

Am incercat ca trecerea dintre straturi sa fie relativ uniforma, alegand astfel sa fie pe puteri consecutive ale lui 2. Fiecare astfel de strat a fost urmat de funcția de activare LeakyReLU, care în urma încercărilor s-a dovedit a fi cea mai adecvata pentru ca evita problema neuronilor morți și mi-a oferit o acuratețe mai buna. De asemenea am introdus și un strat de dropout = 0.3 pentru a preveni overfitting-ul.

ÎNCERCĂRI ÎNREGISTRATE ÎN CADRUL DEZVOLTĂRII MODELULUI MLP

1. Arhitectură de bază

Prima versiune a modelului a avut o structură simplă, formată din trei straturi fully-connected:
Input → 1024 → 512 → 256 → Output (5 clase)

Funcția de activare folosită a fost ReLU, imaginile au fost convertite în grayscale (1 canal) și normalizate în intervalul [-1, 1].

2. Rezultate inițiale (grayscale, 1 canal)

- 20 epoci: 47.93% (Kaggle)
- 35 epoci: 47.66% – am constatat overfitting (rețea prea sensibilă, scor mai slab)
- 10 epoci: 49.40% – cel mai bun scor din această configurație

3. Revenirea la RGB (3 canale)

Analizând vizual imaginile, am realizat că multe detalii relevante se pierd prin conversia în grayscale. Astfel, am păstrat aceleași layere, dar am revenit la imaginile RGB, menținând normalizarea. Scorul a crescut considerabil:

- 56.32% (Kaggle)

4. Teste cu adâncirea arhitecturii

Am testat apoi rețele mai adânci, cu 5 straturi ascunse, dar fără modificări în hiperparametri. Obiectivul era să văd dacă rețeaua poate învăța mai bine cu o capacitate crescută. Din păcate, rezultatele au fost mai slabe:

Arhitectură	Accuracy Kaggle
1024 → 512 → 256 (3 straturi)	56.32%
1024 → 512 → 256 → 128 → 64	54.48%
512 → 256 → 128 → 64 → 32	52.72%

Adâncirea rețelei nu a adus îmbunătățiri, dimpotrivă. Este foarte probabil ca rețeaua să fi intrat în overfitting sau sub învățare, mai ales că nu am ajustat learning rate-ul, batch size-ul sau strategia de regularizare.

5. Îmbunătățiri aproape finale aduse arhitecturii MLP

M-am întors la structura cu 3 straturi, dar am adăugat câteva optimizări:

- Redimensionarea imaginilor la 128x128 px (pentru uniformizare și mai multă informație)
- Dropout (p=0.3) între ultimul hidden layer și stratul de ieșire – pentru regularizare
- Momentum = 0.9 la optimizatorul SGD – pentru învățare mai stabilă și rapidă

→ Rezultatul: 56.72%

6. Experimente adiționale

- Adam în loc de SGD → scorul a scăzut: 55.04%
- Revenire la SGD cu o arhitectură 1024 → 512 → 256 → Output → scor: 59.44%
- Leaky ReLU + scheduler (15 epoci, batch size 64): 59.68% (pe Kaggle: 61.80%)
- Batch size 32 (în loc de 64) a dus la o ușoară scădere: 59.76%

7. Preluarea augmentării CNN și aplicarea la MLP

După ce am observat o creștere semnificativă a performanței în cazul CNN-ului datorită augmentării imaginilor (flip orizontal și rotație), am decis să testez aceeași strategie și pentru MLP. Chiar dacă MLP-urile nu pot capta relații spațiale precum CNN-urile, am vrut să verific dacă îmbunătățirea calității și diversității datelor de intrare poate avea un impact pozitiv.

Am păstrat structura MLP-ului cu 3 straturi complet conectate (1024 → 512 → 256 → 5), dar am aplicat același lanț de transformări folosit anterior:

```
transforms.Resize((128, 128))
transforms.RandomHorizontalFlip(p=0.5)
transforms.RandomRotation(degrees=15)
transforms.ToTensor()
transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
```

Rezultat:

Performanța pe Kaggle a crescut de la aproximativ 61.8% la ~65%, un salt notabil pentru un MLP, dovedind încă o dată că augmentarea imaginilor are un impact benefic chiar și în arhitecturi mai simple, care nu exploatează structura spațială a datelor.

ANALIZA PERFORMANȚEI ÎN FUNCȚIE DE HIPERPARAMETRII TESTAȚI

1. **NUMĂRUL DE EPOCI**

NUMĂR EPOCI	STRUCTURA MLP	CANALE CULOARE	ACURATEȚE
10	1024 → 512 → 256	1	0.49400
20	1024 → 512 → 256	1	0.47933
35	1024 → 512 → 256	1	0.47666

Epocile în plus (deși cu acelea am început) au dus la overfitting → scor mai slab

2. **NUMĂRUL STRATURI**

STRUCTURA MLP	ACURATEȚE
512 → 256 → 128	56.32 %
1024 → 512 → 256 → 128 → 64	54.48%
512 → 256 → 128 → 64 → 32	52.72%

Adâncirea rețelei a dus la scăderea performanței, probabil din cauza overfittingului sau capacității prea mari pentru setul dat.

3. **OPTIMIZATORI**

OPTIMIZATOR	STRUCTURA MLP	NUMĂR EPOCI	ACURATEȚE
SGD	1024 → 512 → 256	10	59.44%
ADAM	1024 → 512 → 256	10	55.04%

Optimizatorul SGD cu momentum s-a dovedit mai stabil și performant în acest caz decât Adam.

4. **BATCH SIZE**

BATCH SIZE	NUMĂR EPOCI	ACURATEȚE
64	10	60.32%
32	10	59.76%

Batch size mai mic a redus performanța ușor – probabil mai mult zgomot în gradient descent.

5. FUNCȚIA DE ACTIVARE & SCHEDULER

FUNCȚIA DE ACTIVARE & SCHEDULER	NUMĂR EPOCI	ACURATEȚE VALIDARE	ACURATEȚE KAGGLE
ReLU (fără scheduler)	10	56.72%	~57.00%
LeakyReLU + scheduler	10	60.32%	61.80%
LeakyReLU + scheduler	15	59.68%	-

Trecerea la LeakyReLU și introducerea unui scheduler adaptiv au crescut performanța semnificativ. Schedulerul a permis modelului să învețe rapid inițial și apoi să stabilizeze antrenarea, evitând să „sară" peste minime bune.

6. CANALE DE CULOARE

FUNCȚIA DE ACTIVARE & SCHEDULER	ACURATEȚE KAGGLE	OBSERVAȚII
1 canal → grayscale	49.40%	Se pierdeau detalii vizuale esentiale
LeakyReLU + scheduler	56.32%	Performanță mai buna datorita informației de culoare

Concluzie MLP

Această etapă a fost esențială pentru înțelegerea limitărilor, dar și a potențialului unui MLP în sarcini de clasificare a imaginilor. Deși inițial performanța s-a plafonat în jurul valorii de 61.8%, testele sistematice cu adâncirea arhitecturii, aplicarea de Dropout, modificarea batch size-ului, dar mai ales introducerea augmentării și preprocesării inspirate din CNN, au dus la o creștere semnificativă a performanței până la aproximativ 65% pe leaderboard-ul Kaggle.

Aceste rezultate demonstrează că, deși MLP-ul nu poate exploata structura spațială a imaginilor ca un CNN, el poate totuși beneficia de o preprocesare atentă și de un set de date diversificat. În final, această experiență a constituit o bază solidă pentru tranziția către rețele convolutionale, evidențiind clar avantajele acestora în procesarea imaginilor, dar și importanța experimentării și rafinării continue.

ARHITECTURA 2 : CNN (Convolutional Neural Network)

CNN-urile sunt rețele neuronale specializate pentru procesarea datelor cu structură spațială (ex: imagini), în care straturile convoluționale învață automat filtre pentru extragerea caracteristicilor spațiale, precum muchii, colțuri sau forme.

După ce am constatat limitările arhitecturii MLP în sarcina de clasificare a imaginilor, am decis să explorez rețelele neuronale convolutionale (CNN), care sunt mai potrivite pentru extragerea automată a caracteristicilor spațiale. Această tranziție s-a dovedit esențială pentru îmbunătățirea performanței.

ARHITECTURA FINALĂ:

Modelul CNN a fost implementat în urma mai multor experimente, iar arhitectura aleasă combina profunzimea rețelei, cu tehnici de regularizare și normalizare, oferind în final cea mai buna performanta de până acum.

Arhitectura acestui model este următoarea:

- 4 straturi convolutionale: 3 → 16 , 16 → 32, 32 → 64, 64 → 128
- Fiecare strat convolutional este urmat de : Batch Normalization, funcție de activare ReLU, MaxPooling (2x2, stride = 2)
- Un strat de dropout de 30%
- 2 straturi fully connected : 128x8x8 → 256 , 256 → 5
- Optimizator SGD cu momentum = 0.9
- Număr de epoci = 60

ÎNCERCĂRI ÎNREGISTRATE ÎN CADRUL DEZVOLTĂRII MODELULUI CNN

1. Prima versiune CNN – Arhitectură de bază

Am implementat o rețea simplă CNN cu 3 straturi convoluționale (Conv2D), fiecare urmat de activare ReLU și MaxPooling:

- Arhitectură: Conv2D(3→16) → MaxPool → Conv2D(16→32) → MaxPool → Conv2D(32→64) → MaxPool → FC(256)
- Dropout = 0.3 după ultimul strat convoluțional
- Preprocesare: imagini RGB, redimensionate la 128x128, normalizate
- Rezultat (15 epoci):
    - VSCode: 82.08%
    - Kaggle: 84.33%

2. Adăugare de augmentare

Am aplicat argumentari simple, dar eficiente:

transforms.RandomHorizontalFlip(p=0.5)
transforms.RandomRotation(degrees=15)

Acestea au introdus variații naturale ale imaginilor, crescând robustetea modelului. Arhitectura CNN a rămas aceeași.

- Rezultat: scorul s-a menținut la 84.33% pe Kaggle, confirmând eficiența augmentării.

3. Extinderea stratului Fully Connected

Am experimentat o rețea cu 4 layere fully connected în loc de 2 (FC: 256 → 128 → 64 → 5 clase), în ideea de a învăța reprezentări mai abstracte.

- Rezultat:
    - VSCode: 79.44%

    - Kaggle: 80.00%

- Concluzie: adâncirea rețelei fully connected nu a adus beneficii; dimpotrivă, a redus scorul – probabil din cauza overfitting-ului sau a pierderii informației prin prea multe transformări dense.

4. Adăugarea unui al 4-lea strat convoluțional

Am crescut complexitatea modelului în zona în care CNN-urile sunt cele mai eficiente: straturile convoluționale.

- Arhitectură nouă: Conv2D(3→16) → Conv2D(16→32) → Conv2D(32→64) → Conv2D(64→128)
- După fiecare: activare ReLU și MaxPooling
- Preprocesare: augmentare + normalizare
- FC layers: doar două (256 → 5 clase)
- Rezultat (30 epoci):
    - VSCode: 83.00%
    - Kaggle: 86.00% (cel mai bun scor atins)

Această arhitectură a învățat ierarhii mai profunde de caracteristici vizuale, oferind modelului o înțelegere mai bună a detaliilor din imagini. Augmentarea și dropout-ul au prevenit overfitting-ul, iar training-ul extins (30 epoci) a permis o convergență mai bună.

5. Încercare de optimizare cu Adam

Din documentare, am aflat că optimizatorul Adam are adesea performanțe mai bune în rețele CNN, în special când datele sunt augmentate. Am înlocuit SGD cu Adam în aceeași arhitectură CNN de 4 straturi.

- Rezultat:
    - Scorul a scăzut drastic – de la 86% la sub 80%
- Concluzie: în acest context, SGD cu momentum + scheduler a fost o alegere mai robustă. Adam a produs instabilitate sau a dus la învățare prea agresivă.

6. Încercare nereușită: adăugare ColorJitter în augmentare

După rezultatele încurajatoare obținute cu augmentări simple (flip orizontal + rotație), am decis să testez o augmentare mai agresivă: ColorJitter, care modifica luminozitatea, contrastul și saturația imaginilor.
Implementare:

transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2)

Aceasta a fost adăugată în lanțul transforms.Compose, în ideea că ar putea introduce o variație suplimentară benefică în imagini.
Rezultat: VSCode (validare): ~80%

În loc să ajute, augmentarea prin ColorJitter a creat variații prea mari față de distribuția inițială a datelor, ceea ce a dus la scăderea capacității modelului de a generaliza corect. Acest experiment a arătat că uneori, augmentarea excesivă poate ruina, așa ca am abandonat aceasta idee.

7. Îmbunătățire finală: Batch Normalization după fiecare strat convoluțional

După ce am testat diverse arhitecturi și augmentări, am continuat documentarea despre rețele CNN pentru a găsi tehnici avansate care pot accelera învățarea și stabiliza antrenamentul.

În urma cercetării mele (inclusiv documentația oficială PyTorch[2] , articole explicative precum cele de pe GeeksForGeeks[3] și lucrarea științifică originală a lui Ioffe și Szegedy[4]), am descoperit că Batch Normalization este o tehnică standard în rețelele neuronale convolutionale moderne, fiind utilizată pentru a accelera antrenamentul și a stabiliza activările intermediare.

BatchNorm normalizează activările unui strat pe mini-batch-uri, aducându-le la o distribuție mai stabilă (medie 0, deviație 1).Acest lucru duce la antrenamente mai rapide, mai stabile și reduce sensibilitatea la inițializarea greșită sau la learning rate prea mare.

Implementare:

Am introdus nn.BatchNorm2d() imediat după fiecare strat convoluțional. Am folosit nn.Sequential pentru a grupa fiecare bloc: Conv2d → BatchNorm → ReLU.

```
self.primul_layerCONV = nn.Sequential(
        # Preia imaginea RGB (3 canale de culoare), aplica 16 filtre 3x3 și normalizeaza activarile
        nn.Conv2d(3, 16, kernel_size=3, padding=1),
        nn.BatchNorm2d(16), # BatchNorm2d stabilizeaza invatarea,scade riscul de overfitting si normalizeaza valorile activarii
        nn.ReLU() # funcția de activare --> elimina valorile negative
    )
```

Rezultat:

- VSCode: 88.88%
- Kaggle: 90%

Această modificare a făcut o diferență semnificativă, mai ales în combinație cu augmentarea moderată și training-ul prelungit (60 epoci), folosind SGD cu StepLR[5].

ANALIZA PERFORMANȚEI ÎN FUNCȚIE DE HIPERPARAMETRII TESTAȚI

1. ARHITECTURA CNN-ULUI

ARHITECTURA CONVOLUTIONALA	FULLY CONNECTED LAYERS	EPOCI	ACURATEȚE
Conv(3 straturi: 16→32→64)	256 → 5	15	84.33%
Conv(3 straturi: 16→32→64)	256 → 128 → 64 → 5	15	80.00%
Conv(4 straturi: 16→32→64→128)	256 → 5	15	83.00%

Epocile în plus (deși cu acelea am început) au dus la overfitting → scor mai slab

[2] PyTorch Documentation – `BatchNorm2d`. Disponibil la:https://docs.pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html

[3] GeeksForGeeks - `What is Batch Normalization In Deep Learning?` . Disponibil la: https://www.geeksforgeeks.org/what-is-batch-normalization-in-deep-learning/

[4] Ioffe, S., & Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. In Proceedings of the 32nd International Conference on Machine Learning (ICML). PDF: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://arxiv.org/pdf/1502.03167

[5] PyTorch Documentation – Learning Rate Scheduling (`torch.optim.lr_scheduler`). Disponibil la: https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate

2. NUMĂRUL DE EPOCI

ARHITECTURA CONVOLUTIONALA	EPOCI	ACURATEȚE
Conv(3 straturi) + FC(256→5)	15	84.33%
Conv(4 straturi) + FC(256→5)	15	83.00%
Conv(4 straturi) + FC(256→5)	30	86.00%
Conv(4 straturi) + FC(256→5)	60	~90.80%

În acest caz, creșterea numărului de epoci a adus o îmbunătățire clară – spre deosebire de comportamentul observat la MLP, unde mai multe epoci au dus la overfitting.Astfel în final am ajuns la un număr de epoci egal cu 60.

3. MODIFICĂRI ARHITECTURALE + MODIFICĂRI DE ANTRENARE

MODIFICARE PRINCIPALA	DETALII	EPOCI	ACURATEȚE VSCODE	ACURATEȚE KAGGLE
Arhitectură simplă (3 conv + 1 FC)	Dropout=0.3, RGB, normalizare	15	82.08%	84.33%
Adăugare augmentare (flip + rotație)	Fără schimbări arhitecturale	15	82.08%	84.33%
Extindere FC layers (256→128→64→5)	Fără schimbări în conv	15	79.44%	80.00%
Adăugare conv4 (3→16→32→64→128)	FC(256→5), augmentare	15	83.00%	83.00%
Training la 30 epoci	Arhitectura anterioara	30	83.00%	86.00%
Optimizator: Adam (în loc de SGD)	Pe modelul cu conv4	30	~78.00%	<80.00%
Augmentare agresivă: ColorJitter	brightness, contrast, saturation	30	~80.00%	-
Adăugare BatchNorm după fiecare strat conv	Conv + BN + ReLU (Sequential)	60	88.84%	90.80%

Concluzie CNN

Modelul final a folosit o arhitectură CNN cu 4 straturi convoluționale, fiecare urmat de **Batch Normalization**, activare ReLU și MaxPooling. Am combinat această arhitectură cu augmentare moderată (flip orizontal și rotație), normalizare, Dropout și optimizare cu **SGD + scheduler**.

Această combinație a dus la **cea mai performantă versiune** testată:

- **Acuratețe VSCode**: 88.64%
- **Acuratețe finală Kaggle**: **89.86%**

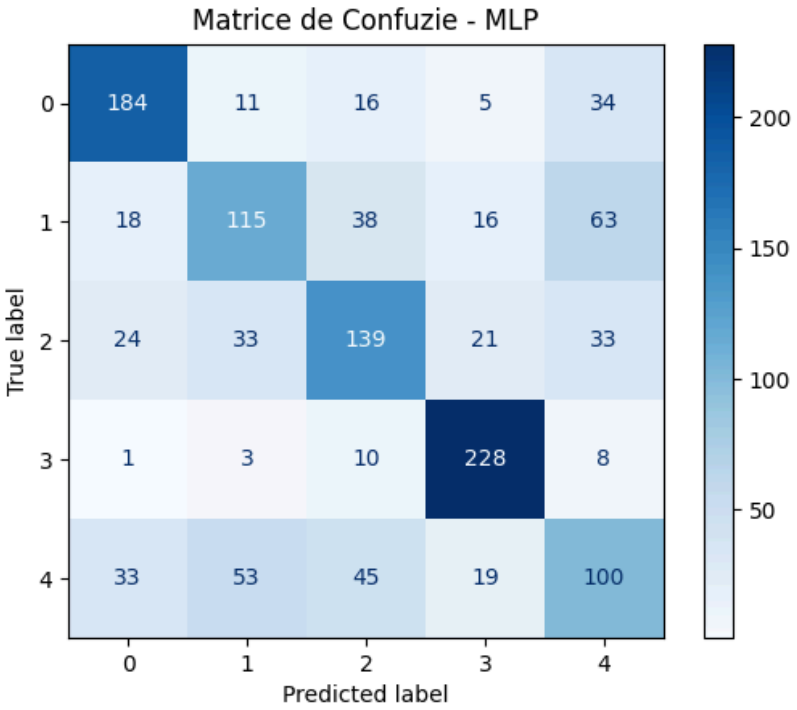
Performanța este semnificativ mai mare comparativ cu modelele MLP testate anterior (diferență de peste 25 de procente), dovedind clar avantajele arhitecturilor CNN în procesarea imaginilor.

În plus, adăugarea stratului BatchNorm2d s-a dovedit a fi un pas cheie în îmbunătățirea stabilității și a vitezei de antrenare, conform celor învățate în etapa de documentare teoretică.

Această tranziție de la MLP la CNN, urmată de o rafinare atentă a arhitecturii și hiperparametrilor, a marcat un **salt calitativ în învățarea automată aplicată** și a condus la un scor competitiv în clasamentul final.

MATRICI DE CONFUZIE. COMPARAREA CELOR 2 MODELE.

Matricea de confuzie a modelului MLP :



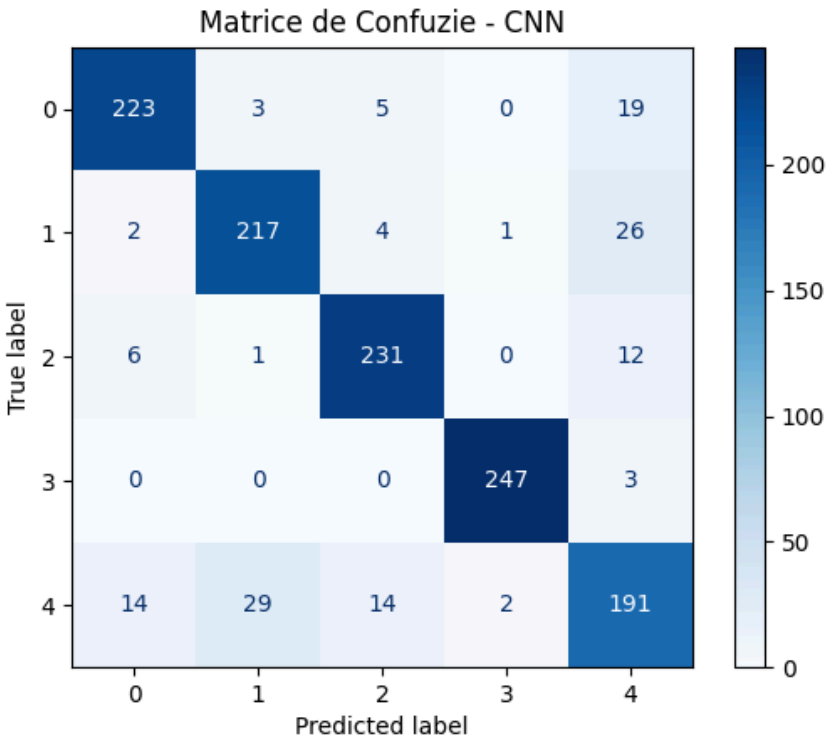
INTERPRETARE MATRICE DE CONFUZIE → MLP

CLASA	CLASIFICARI CORECTE	CONFUZII	OBSERVATII
0	184	clasa 4(34),clasa 2(16), clasa 1(11),clasa 3(5)	Confuzie frecventă cu clasele 2 și 4 ,dar aproape deloc cu clasa 3
1	115	clasa 4(63),clasa 2(38), clasa 0(18),clasa 3(16)	Se confunda des cu clasa 4 – separabilitate slabă
2	139	clasa 1(33),clasa 4(33), clasa 0(24),clasa 3(21)	Ambiguitate moderată – multe confuzii dispersate
3	228	clasa 2(10),clasa 4(8), clasa 1(3),clasa 0(1)	Cea mai bine clasificată – trăsături distinctive
4	100	clasa 1(53),clasa 2(45), clasa 0(33),clasa 3(19)	Confuzii severe cu clasele 1 și 2 – greu de distins

Modelul MLP se descurcă foarte bine pe clasa 3, ceea ce arată că dacă datele sunt clar diferențiabile, chiar și un MLP poate performa decent.Cele mai mari probleme apar între clasele 1 și 4, dar și între 2 și restul. Asta confirmă limitele MLP-ului când vine vorba de interpretat informație spațială din imagini.

Această analiză evidențiază necesitatea utilizării unor arhitecturi spațiale, precum CNN, în sarcinile de clasificare a imaginilor cu patternuri complexe

Matricea de confuzie a modelului CNN :



INTERPRETARE MATRICE DE CONFUZIE –> CNN

CLASA	CLASIFICARI CORECTE	CONFUZII	OBSERVAȚII
0	223	clasa 4(19),clasa 2(5), clasa 1(3),clasa 3(0)	Confuzii minore cu clasa 4 – dar în general, modelul o recunoaște bine.
1	217	clasa 4(26),clasa 2(4), clasa 0(2),clasa 3(1)	Foarte clară, dar uneori confundă cu clasa 4.
2	231	clasa 4(12),clasa 0(6), clasa 1(1),clasa 3(0)	Similar cu clasa 1 – clasa 4 e mai problematică.
3	247	clasa 4(3),clasa 2(0), clasa 1(0),clasa 0(0)	Cea mai bine clasificată – caracteristici clare.
4	191	clasa 1(29),clasa 2(14), clasa 0(14),clasa 3(2)	Clasa cea mai ambiguă – confundată cu toate celelalte.

Analizând această matrice de confuzie, devine evident că modelul CNN aduce o îmbunătățire substanțială față de MLP. Toate clasele sunt clasificate mai precis, cu mai puține confuzii, iar distribuția greșelilor este mult mai concentrată.

Clasa 3 rămâne foarte bine recunoscută, dar acum și clasele 0, 1 și 2 sunt aproape perfect separate.

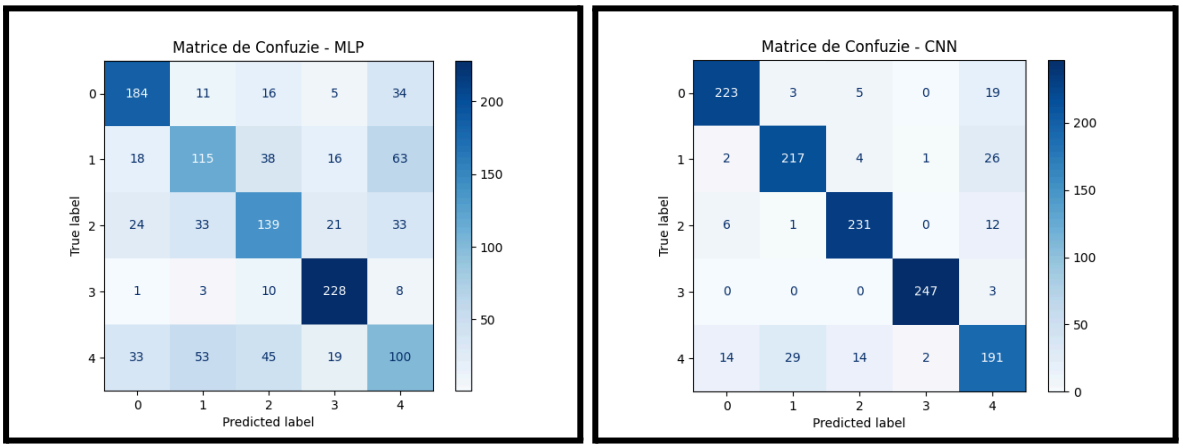
Singura clasă care ridică în continuare dificultăți este clasa 4, care tinde să fie confundată cu clasele apropiate vizual – o problemă comună în clasificarea imaginilor.

Această performanță arată clar forța rețelelor convolutionale în învățarea caracteristicilor spațiale relevante direct din imagini, fără extragere manuală. În plus, faptul că am utilizat augmentare moderată și Batch Normalization a contribuit decisiv la stabilitatea și generalizarea modelului.

CNN-ul s-a dovedit astfel o alegere mult mai potrivită pentru această sarcină, fiind capabil să surprindă subtilități și patternuri vizuale complexe, pe care un MLP le ignoră complet.

COMPARARE:

Pentru o mai bună evidențiere a diferenței de performanță, am alăturat matricile de confuzie ale celor două modele:



Se observă clar cum modelul CNN are diagonala mult mai pronunțată (clasificări corecte), în timp ce MLP-ul are multe confuzii răspândite în afara acesteia. În special:

- **MLP** face multe confuzii între clasele apropiate (1 vs 4, 2 vs 3), cu un model care nu înțelege relațiile spațiale.
- **CNN**, în schimb, reușește să izoleze mult mai bine fiecare clasă, chiar și în cazul celor mai greu de distins (ex: clasa 4).

Această vizualizare dublă sprijină observația anterioară: arhitectura CNN este net superioară MLP-ului în sarcini de clasificare a imaginilor, datorită capacității sale de a învăța reprezentări complexe și ierarhice.

CONCLUZIE GENERALA

Acest proiect a reprezentat pentru mine o oportunitate excelentă de a învăța și în același timp de a experimenta concret tehnici esențiale din domeniul învățării automate aplicate pe imagini. Am început cu un model MLP simplu, inspirat din laboratoarele anterioare, iar pe parcurs, printr-un proces de explorare, testare și documentare, am ajuns să construiesc și să optimizez o rețea convoluțională care să ofere rezultate semnificativ mai bune.

Am învățat că alegerea corectă a arhitecturii nu este singura cheie a succesului. Preprocesarea imaginilor, tehnicile de augmentare, activarile utilizate, precum și hiperparametrii precum numărul de epoci, batch size-ul, optimizatorul sau dropout-ul influențează major performanța finală. Fiecare modificare a fost motivată, testată și comparată în mod riguros, ceea ce mi-a consolidat înțelegerea asupra modului în care fiecare componentă a pipeline-ului contribuie la rezultatul final.

Privind în urmă, dacă aș relua proiectul, poate aș acorda mai multă atenție inițială unor metode de extragere de caracteristici vizuale sau aș încerca metode mai avansate de regularizare. Totodată, voi continua să explorez importanța alegerii corecte a optimizatorului și a strategiilor de learning rate scheduling. Per ansamblu, experiența dobândită mi-a crescut considerabil încrederea în lucrul cu rețele neuronale și mi-a arătat cât de mult contează să experimentezi, să greșești, și să înveți iterativ.