

Tema 2 - Machine Learning

Fichios Mara

342 C3

mara.fichios@stud.acs.upb.ro

Rezumat

Această lucrare investighează clasificarea de imagini în două regimuri vizuale distincte: clasificare de obiecte în setul *Imagebits* și clasificare în setul *Land patches*. Pentru ambele sarcini urmărим un flux complet, de la analiza exploratorie a datelor (echilibrul claselor, variabilitate intra-/inter-clasă, verificări de integritate și exemple vizuale), până la proiectarea și antrenarea unor modele neurale de bază (MLP) și convolutionale (CNN). Ne concentrăm pe factorii care influențează generalizarea: normalizarea intrărilor (estimarea statisticilor canalelor), alegerea hiperparametrilor (optimizator, rată de învățare, regularizare), precum și efectul augmentărilor (transformări geometrice și fotometrice) asupra curbelor de antrenare și a performanței finale. Evaluarea se face prin metrii standard (acuratețe, macro-F1) și prin analiza matricilor de confuzie, pentru a înțelege tiparele de eroare și clasele problematice. În plus, pentru *Land patches* discutăm impactul inițializării și al transferului între seturi (fine-tuning de la un model antrenat pe *Imagebits*), ca metodă de a îmbunătăți stabilitatea și performanța în scenarii cu date mai dificile.

1 Introducere

Clasificarea de imagini este o problemă fundamentală în *computer vision*, cu aplicații în recunoașterea obiectelor, automatizări industriale și analiză geospațială. Deși la nivel conceptual sarcina este aceeași (asocierea unei etichete unei imagini), dificultatea practică depinde puternic de natura datelor: cât de consistente sunt exemplele din aceeași clasă, cât de mari sunt variațiile de iluminare/pozitie/scără și cât de mult se suprapun vizual clasele între ele. În acest context, o soluție robustă necesită atât o arhitectură potrivită, cât și un *setup* de antrenare atent ales (normalizare, regularizare, augmentări, selecția hiperparametrilor).

În această lucrare analizăm două seturi de date cu proprietăți complementare. *Imagebits* conține imagini RGB de rezoluție mică (96×96) cu 10 clase de obiecte (ex. avion, mașină, câine), unde variațiile tipice includ schimbări de fundal, pozitie și iluminare. În contrast, *Land patches* este construit din imagini de la sateliți (64×64 , RGB) pentru clase de acoperire a terenului (ex. pădure, autostradă, rezidențial), unde separarea claselor poate depinde de tipare fine și de structuri repetitive. Această diferență de domeniu conduce la provocări distincte: un MLP pe pixeli poate funcționa ca un baseline, dar în general nu exploatează invarianta spațială, în timp ce un CNN poate învăța filtre locale și ierarhii de caracteristici, fiind mai adecvat pentru date vizuale.

2 Analiza setului de date Imagebits

2.1 Descrierea setului de date și analiza exploratorie

2.1.1 Maparea etichetelor la nume de clasă

Setul *Imagebits* conține 10 clase, etichetate inițial numeric (string-uri în metadate). Pentru lizibilitate și interpretare am definit un dicționar de mapare `id_to_name`, astfel încât fiecare label să fie asociat cu o denumire semantică (ex. "1" → *airplane*). Această mapare este folosită

ulterior atât în vizualizări, cât și în rapoarte de performanță (matrici de confuzie, classification report).

Tabela 1: Maparea label → nume clasă în *Imagebits*.

Label	Clasă
1	airplane
2	bird
3	car
4	cat
5	deer
6	dog
7	horse
8	monkey
9	ship
10	truck

2.1.2 Echilibrul claselor și dimensiunea seturilor

Am analizat distribuția etichetelor separat pentru `train` și `test`. Rezultatele arată un set perfect echilibrat: fiecare clasă are 800 imagini în `train` și 500 imagini în `test`. Prin urmare, metrii precum acuratețea sunt relevante (nu sunt distorsionate de dezechilibre), iar selecția hiperparametrilor nu necesită ponderi de clasă. În total, setul conține 8000 imagini pentru antrenare și 5000 pentru testare.

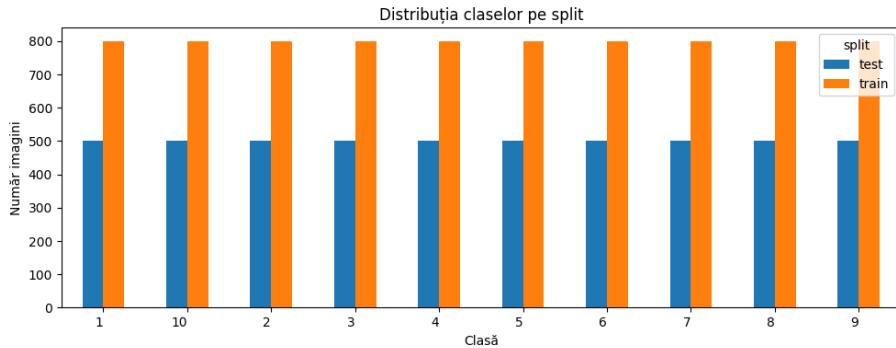


Figura 1: Distribuția claselor pe split (`train` vs `test`) în *Imagebits*.

2.1.3 Verificări de integritate

Înainte de EDA și modelare am făcut verificări de integritate pentru a identifica imagini corupte sau inconsistente ca dimensiune/format. Pe un eșantion din date am obținut:

- **Erori la citire:** 0 (nu există imagini corupte în eșantionul verificat);
- **Dimensiuni unice:** (96, 96) (toate imaginile au aceeași rezoluție);
- **Moduri unice:** RGB (toate imaginile sunt color, 3 canale).

Această consistență simplifică pipeline-ul: nu este necesară redimensionare dinamică sau tratare separată pentru imagini grayscale.

2.1.4 Variabilitate intra-clasă și separabilitate inter-clasă

Am vizualizat exemple aleatorii din `train` pentru fiecare clasă pentru a evalua variabilitatea vizuală și posibile surse de confuzie. Se observă **variabilitate intra-clasă** ridicată: obiectele apar la scări diferite, în poziții diferite, cu fundaluri diverse și iluminare variabilă (ex. *airplane* fotografiat în aer vs. pe pistă; *truck* în medii urbane vs. șosea).

În același timp, există și **suprapunerile inter-clasă** la nivel de context: clase precum *airplane* și *ship* apar frecvent cu fundal albastru (cer/apă), iar clase de animale (*deer*, *horse*) pot avea texturi și culori similare în scene naturale. Aceste observații justifică utilizarea unui model convecțional (CNN), care poate învăța pattern-uri locale robuste (margini, texturi) și invariante la translație/scala obiectului.

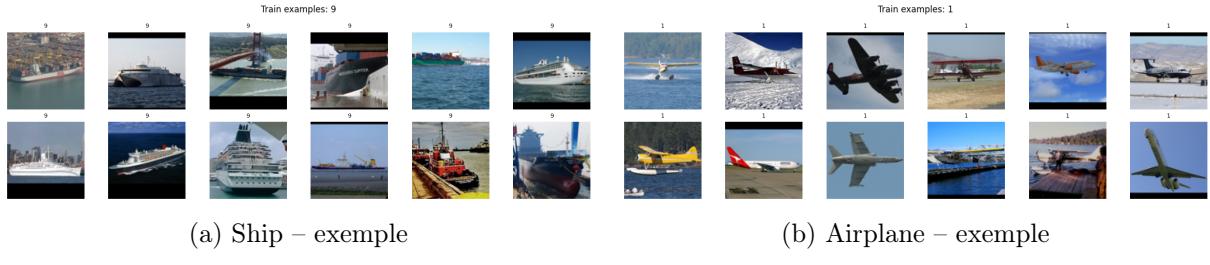


Figura 2: Exemple vizuale pentru clasele *ship* și *airplane*. Se observă fundaluri frecvent albastre (apă/cer), care pot introduce similitudini cromatice și pot crește riscul de confuzie între clase.

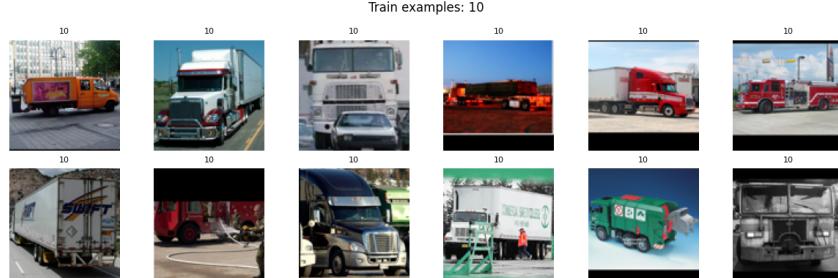


Figura 3: Truck - exemple

2.1.5 Statistici globale pe canale (mean/std) pentru normalizare

Pentru a stabili o normalizare corectă a intrărilor (standardizare pe canale), am calculat media și deviația standard pe canalele RGB folosind toate imaginile din `train`. Am folosit o metodă de tip *streaming* (acumulare a sumei și a sumei pătratelor), care evită încărcarea simultană a întregului set în memorie.

Rezultatele obținute sunt: $\mu = [0.4472, 0.4396, 0.4050]$, $\sigma = [0.2606, 0.2567, 0.2700]$ (valori în intervalul [0,1], după scalarea pixelilor la 255).

Aceste statistici sunt folosite ulterior în transformările de preprocesare, prin standardizarea fiecărei imagini:

$$x' = \frac{x - \mu}{\sigma}.$$

Tabela 2: Statistici globale RGB pe `train` (*Imagebits*).

	R	G	B
mean (μ)	0.4472	0.4396	0.4050
std (σ)	0.2606	0.2567	0.2700

2.1.6 Statistici cromatice per clasă

Pentru a înțelege dacă anumite clase au distribuții cromatice distincte (bias de fundal), am calculat media și deviația standard RGB **separat pe fiecare clasă**, folosind un eșantion de până la 400 imagini/clasă (pentru eficiență). Tabelul 3 indică urmatoarele diferențe: clase precum *airplane* și *ship* au în medie valori mai mari pe canalul B (fundal cer/apă), în timp ce clasele de animale tind să fie mai „calde” (R/G mai mari) datorită fundalurilor naturale.

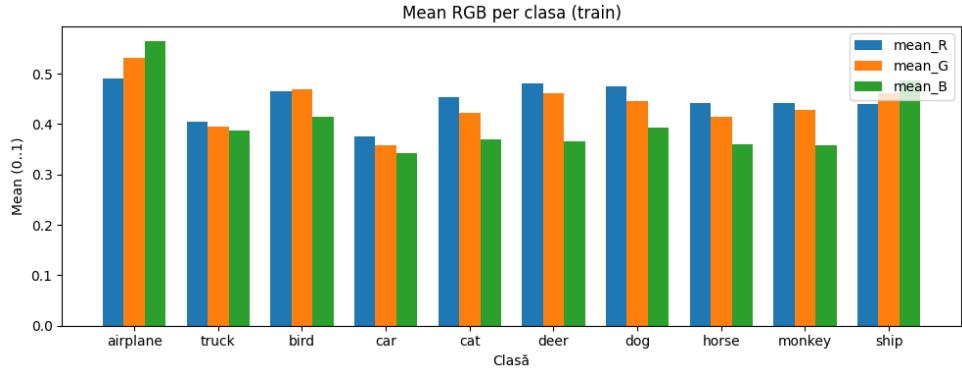


Figura 4: Media canalelor RGB per clasă (train).

Tabela 3: Statistici RGB per clasă (train).

Clasă	mean_R	mean_G	mean_B	std_R	std_G	std_B
airplane	0.4908	0.5321	0.5649	0.2505	0.2461	0.2737
truck	0.4042	0.3947	0.3880	0.2963	0.2953	0.3067
bird	0.4649	0.4684	0.4147	0.2431	0.2381	0.2688
car	0.3748	0.3577	0.3422	0.2902	0.2778	0.2816
cat	0.4538	0.4231	0.3696	0.2474	0.2389	0.2373
deer	0.4814	0.4622	0.3651	0.2248	0.2160	0.2164
dog	0.4750	0.4463	0.3933	0.2585	0.2507	0.2542
horse	0.4425	0.4154	0.3591	0.2536	0.2506	0.2495
monkey	0.4417	0.4281	0.3580	0.2451	0.2381	0.2362
ship	0.4395	0.4612	0.4856	0.2741	0.2751	0.2900

2.1.7 Indice cromatic: dominanța albastrului

Pentru a sumariza rapid diferențele de culoare între clase am definit un indice cromatic simplu:

$$\Delta_{blue} = \text{mean}_B - \text{mean}_R,$$

calculat pe imaginile din **train**. Valori pozitive sugerează o componentă albastră dominantă (cer/apă), iar valori negative sugerează tonuri mai „calde” sau scene terestre.

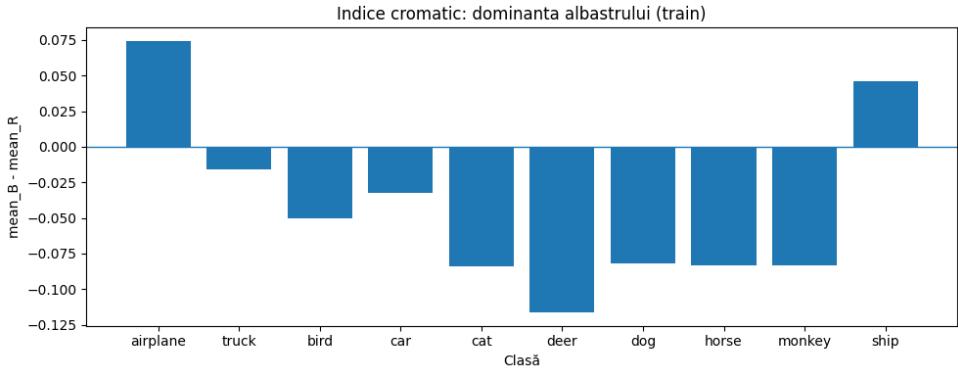


Figura 5: Indice cromatic Δ_{blue} per clasă (train).

Se observă că *airplane* și *ship* au cele mai mari valori, ceea ce este consistent cu apariția frecventă a cerului și a apei în fundal. Majoritatea claselor de animale au valori negative, indicând fundaluri naturale (pământ/vegetație) și obiecte cu texturi mai puțin albastre. Acest tip de bias de fundal poate contribui la confuzii între clase cu contexte similare și motivează augmentări care reduc dependența de culoare.

2.1.8 Distribuția luminozității: comparație train vs test

Am analizat distribuția luminozității (brightness) pentru un eșantion de imagini din **train** și **test**, pentru a verifica dacă există un *shift* între splituri (ex. test mai întunecat / mai luminos).

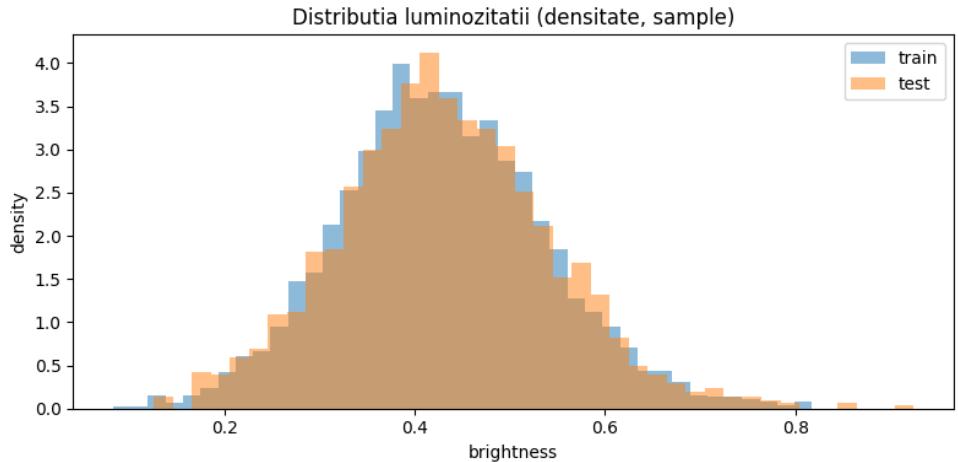


Figura 6: Distribuția luminozitatii pentru **train** și **test**

Distribuțiile se suprapun aproape complet, ceea ce sugerează că **train** și **test** sunt comparabile din punct de vedere fotometric. Prin urmare, performanța pe test nu ar trebui să fie afectată de diferențe sistematice de expunere/iluminare, iar o normalizare globală pe canale este adecvată.

2.1.9 Construirea setului de validare (split stratificat)

Setul *Imagebits* furnizează doar spliturile **train** și **test**, fără un folder dedicat de validare. Pentru selecția hiperparametrilor și controlul overfitting-ului am creat un **validation split** din **train**, folosind **train_test_split** cu: **test_size=0.2**, **random_state** fix și **stratificare după etichetă**.

În urma împărțirii, pentru fiecare clasă au rezultat aproximativ 640 imagini în `train` și 160 în `val` (menținând distribuția perfect echilibrată). Setul de `test` rămâne neschimbat și este folosit doar pentru evaluarea finală.

2.2 Model MLP

2.2.1 Preprocesare și augmentări (Albumentations) motivate de EDA

Din analiza exploratorie pentru *Imagebits* am observat:

- **Variatie intra-clasă semnificativă:** obiectele apar la scară diferite, din unghiuri diferite și pe fundaluri variate;
- **Bias cromatic** pentru anumite clase (ex. *airplane* și *ship*): fundaluri frecvent albastre (cer/apă), care pot induce similarități între clase;
- **Distribuții de luminozitate similare** între `train` și `test`: nu se observă un *domain shift* fotometric major, dar există variații locale de expunere și contrast.

Pe baza acestor observații am definit două pipeline-uri de transformări:

- **fără augmentări (baseline):** doar normalizare pe canale folosind statisticile estimate pe `train`
- **cu augmentări moderate:** transformări care introduc variații plauzibile pentru imagini de obiecte naturale, fără a distruge semnalul semantic:
 - `HorizontalFlip` ($p=0.5$): multe clase sunt aproximativ invariante la oglindire (car, animal, avion), iar flip-ul ajută la generalizare la orientări diferite;
 - `ShiftScaleRotate` (shift 0.05, scale 0.10, rotate 15, $p=0.5$): acoperă variații de poziție în cadru, zoom și rotații mici, coerente cu variabilitatea de compoziție observată în EDA;
 - `ColorJitter` (brightness/contrast/saturation/hue, $p=0.5$): reduce dependența de culoare și iluminare (relevant pentru bias-ul albastru din *airplane/ship* și pentru variații de expunere), forțând modelul să învețe caracteristici mai robuste decât culoarea brută;
 - `Normalize` cu variabilele estimate din `train`, pentru stabilitate numerică și convergență mai bună.

Prin design, augmentările sunt **moderate**: nu folosim rotații mari sau deformări agresive, deoarece la rezoluție mică (96×96) acestea pot degrada rapid semnalul și pot introduce imagini nerealiste.

2.2.2 Dataset și DataLoader pentru MLP

Am implementat o clasă `ImageBitsDataset` care:

1. încarcă imaginea RGB și aplică transformările Albumentations;
2. convertește rezultatul în tensor;
3. pentru MLP, aplică `flatten` la un vector de dimensiune $96 \cdot 96 \cdot 3 = 27648$.

Batch-urile sunt create cu `DataLoader` folosind `batch_size=128`. Pentru o comparație corectă, validarea se face **fără augmentări** (doar normalizare), astfel încât scorul pe `val` să reflecte generalizarea pe date nemodificate.

2.2.3 Arhitectura MLP și limitările modelului

MLP-ul folosit este un baseline care operează direct pe pixeli (fără structură spațială), cu două straturi ascunse:

$$27648 \rightarrow 1024 \rightarrow 512 \rightarrow 10,$$

cu **BatchNorm** și **Dropout** după fiecare strat și activare **ReLU**. BatchNorm stabilizează distribuțiile interne și accelerează antrenarea, iar Dropout regularizează, reducând co-adaptarea neuronilor.

Totuși, un MLP pe pixeli are o limitare majoră: **nu exploatează invarianta la translație** și nu învață filtre locale precum un CNN; vede fiecare pixel ca o caracteristică independentă. În plus, numărul de parametri este foarte mare:

$$\approx 28.85 \text{ milioane parametri},$$

ceea ce crește riscul de overfitting și necesită regularizare/augmentare. Dimensiunea mare a modelului, raportată la informația utilă din imagini la 96×96 , explică de ce un MLP tinde să memorizeze tipare de train în loc să generalizeze.

2.2.4 Setup de antrenare și criteriu de selecție

Am antrenat MLP-ul cu:

- **CrossEntropyLoss** (problemă multi-class, set echilibrat);
- **AdamW** ($lr = 10^{-3}$, $weight_decay = 10^{-4}$) pentru stabilitate și regularizare L2 implicită;
- monitorizare pe **macro-F1** în validare (media F1 pe clase), metrică relevantă pentru evaluare per-clasă;
- **early stopping** cu `patience=7` epoci (criteriu: îmbunătățire `val_f1` cu `min_delta`).

2.2.5 Comparație: MLP fără augmentări vs cu augmentări

Am rulat două antrenări identice (aceeași arhitectură, aceeași hiperparametri), diferind doar prin transformările aplicate în `train`. În Figura ?? sunt afișate curbele pe validare pentru **loss** și **macro-F1**.

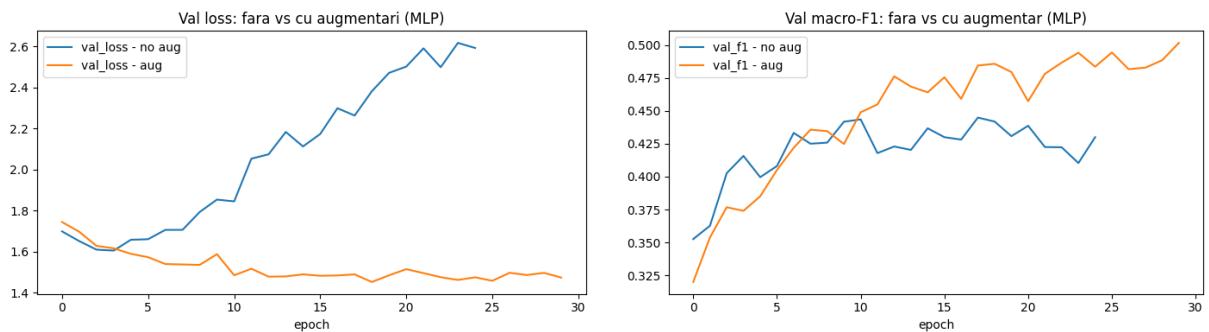


Figura 7: MLP: comparație pe validare între *no-aug* și *with-aug*

Se observă un comportament foarte diferit între cele două setări:

- **Fără augmentări:** `val_loss` crește constant după primele epoci, în timp ce `val_macro-F1` rămâne relativ plafonat (în jur de 0.43–0.45). Acest tipar indică **overfitting puternic**: modelul devine tot mai încrezător pe predicții greșite în validare (cross-entropy penalizează puternic aceste cazuri), chiar dacă deciziile de tip argmax nu se îmbunătățesc semnificativ.

- **Cu augmentări:** `val_loss` rămâne mai mic și mai stabil (în jur de 1.45), iar `val_macro-F1` urcă gradual până la ≈ 0.50 . Augmentările acționează ca o formă de regularizare: cresc diversitatea datelor de train și reduc memorarea fundalurilor/colorilor specifice, fără modelul să învețe reguli mai robuste.

Concluzia acestei comparații este că, pentru un MLP cu mulți parametri, augmentările sunt esențiale pentru a limita overfitting-ul; de aceea, în continuare am păstrat varianta **cu augmentări** ca bază pentru tuning.

2.2.6 Tuning de hiperparametri: reducerea capacitatii și regularizare

Având în vedere numărul mare de parametri și overfitting-ul observat, am încercat:

- **reducerea dimensiunii straturilor ascunse** ($1024/512 \rightarrow 512/256$), pentru a micșora capacitatea și a reduce memorarea;
- **dropout mai mare** ($0.3 \rightarrow 0.4$), pentru regularizare mai puternică;
- ajustarea **learning rate**-ului (10^{-3} vs $3 \cdot 10^{-4}$), pentru stabilitatea optimizării și generализare.

Rezultatele pe validare pentru configurațiile testate (toate cu augmentări în train) au fost:

- **MLP(1024,512, dropout=0.3)**: best `val_F1` ≈ 0.502 (la epoca 30);
- **MLP(512,256, dropout=0.4)**: best `val_F1` ≈ 0.511 (la epoca 44) – **cea mai bună pe validare**;
- **MLP(512,256, dropout=0.4, lr=3e-4)**: best `val_F1` ≈ 0.506 (early stop epoca 36).

Interpretare: reducerea dimensiunii straturilor și creșterea dropout-ului au îmbunătățit generalizarea (macro-F1 mai bun), deoarece modelul are mai puțină capacitate de memorare și este fără să învețe reprezentări mai stabile. În schimb, un *learning rate* prea mic a redus viteza de învățare și nu a adus câștig suplimentar în acest setup.

2.2.7 Evaluare pe test și alegerea modelului final

Modelul selectat pentru testare a fost cel cu cea mai bună performanță pe validare:

MLP(512,256, dropout=0.4) + augmentări.

Pe setul de test, performanța obținută a fost:

`loss = 1.378, accuracy = 0.5058, macro-F1 = 0.5013.`

Acest rezultat este consistent cu validarea și reprezintă cel mai bun compromis dintre configurațiile încercate. Totuși, scorul rămâne relativ redus pentru o problemă cu 10 clase, ceea ce reflectă limita fundamentală a unui MLP pe pixeli: lipsa inductive bias-ului spațial (spre deosebire de CNN) și sensibilitatea la variații de poziție/structură.

2.2.8 Analiza erorilor: matricea de confuzie și performanța pe clase

În Figura 8 este matricea de confuzie pe test. Se observă confuzii puternice între clase vizual apropriate sau cu context similar:

- **animal vs animal:** *cat/dog/deer/horse/monkey* se confundă frecvent, deoarece la rezoluție mică diferențele de textură și formă sunt subtile, iar MLP nu exploatează localitatea;

- **vehicule:** *car* și *truck* sunt relativ bine recunoscute comparativ cu animalele, probabil datorită formelor globale și contextelor (drum/urban) mai consistente;
- **airplane vs ship:** pot apărea confuzii motivate și de bias-ul cromatic (fundal albastru), discutat în EDA.

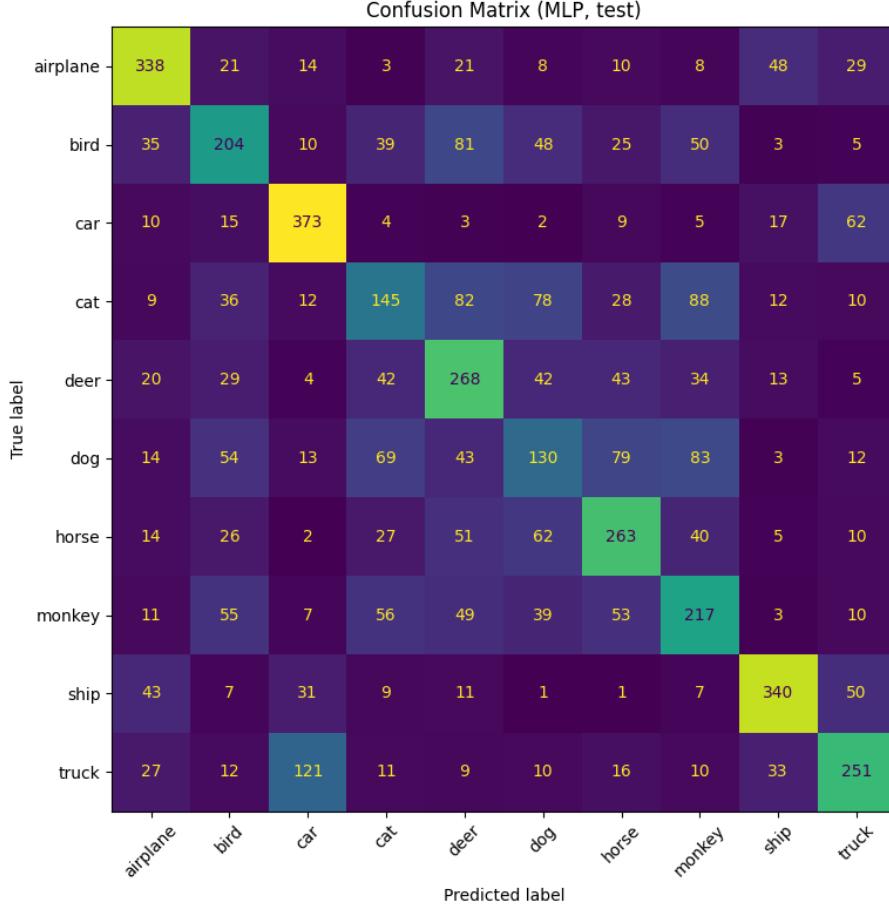


Figura 8: Matricea de confuzie pentru MLP (test).

Pentru o analiză mai bună, am calculat acuratețea pe clasă (support=500 fiecare). Clasele cele mai dificile pentru MLP sunt *dog* și *cat* (sub 0.30), iar cele mai ușoare sunt *car* (0.746), *ship* (0.680) și *airplane* (0.676). Acest tipar este intuitiv: vehiculele au siluete mai consistente, în timp ce animalele variază mult în poză (poziție, fundal, occluderi), iar fără filtre convoluționale modelul nu poate separa bine aceste variații.

Tabela 4: Acuratețe pe clasă pentru MLP (test).

Clasă	acc_per_class	support
dog	0.260	500
cat	0.290	500
bird	0.408	500
monkey	0.434	500
truck	0.502	500
horse	0.526	500
deer	0.536	500
airplane	0.676	500
ship	0.680	500
car	0.746	500

În concluzie, MLP-ul oferă un baseline util pentru a evidenția dificultatea problemei și importanța inductive bias-ului spațial. Augmentările și regularizarea îmbunătățesc vizibil generalizarea, dar pentru performanțe competitive este necesară o arhitectură convoluțională (CNN), care poate învăța caracteristici locale și invariante la transformări.

2.3 Model CNN

2.3.1 Augmentări adiționale pentru CNN: RandomResizedCrop

Pentru CNN am folosit același principiu ca la MLP: **validarea rămâne fără augmentări** (doar normalizare), iar augmentările se aplică doar pe `train`. Motivația augmentărilor:

- **Variatie intra-clasă:** obiectele apar la scări și poziții diferite în cadrul unei același clase;
- **Bias cromatic:** pentru clase precum *airplane* și *ship* apare frecvent fundal albastru (cer/apă), care poate induce similarități între clase;
- **Variatii moderate de iluminare/contrast:** distribuțiile globale sunt similare între `train` și `test` (fără *domain shift* major), dar există diferențe locale de expunere și contrast.

Initial am încercat un **crop aleator cu redimensionare**: `RandomResizedCrop(size=(96, 96), scale=(0.85, 1.0), ratio=(0.9, 1.1))`. Intuiția a fost că modelul să fie robust la **încadrări diferite** ale obiectului, similar cu ce se face frecvent în clasificarea de obiecte.

Totuși, pe un set cu rezoluție mică (96×96) acest tip de crop poate deveni riscant: dacă obiectul ocupă o zonă mică, un crop agresiv poate elimina părți relevante (sau chiar obiectul aproape complet), transformând eticheta într-un semnal ambiguu. În practică, acest lucru s-a reflectat în scăderea performanței pe validare, motiv pentru care am renunțat la crop și am trecut la augmentări **mai sigure**, care păstrează mai bine conținutul semantic: flip, rotații mici și jitter de culoare.

2.3.2 Arhitectura CNN de bază (SimpleCNN) și comparația cu MLP

Prima arhitectură CNN (SimpleCNN) are trei blocuri convoluționale:

$$(3 \rightarrow 32) \rightarrow (32 \rightarrow 64) \rightarrow (64 \rightarrow 128),$$

fiecare cu **Conv-BN-ReLU** urmat de **MaxPool** (reducere spațială $96 \rightarrow 48 \rightarrow 24 \rightarrow 12$), și un **head** simplu cu `AdaptiveAvgPool2d(1) + Linear(128, 10)`.

Un avantaj major față de MLP este numărul de parametri:

$$\text{SimpleCNN: } \approx 0.095M \text{ parametri vs. MLP: } \approx 28.85M.$$

CNN-ul este mult mai compact deoarece folosește **weight sharing** (aceleasi filtre aplicate local pe toată imaginea), și are un **inductive bias** potrivit pentru imagini (filtre locale, invarianta la translație). Prin urmare, ne aşteptăm la o generalizare mai bună chiar cu mai puțini parametri.

Antrenarea folosește același setup ca la MLP: `CrossEntropyLoss`, `AdamW`, și **early stopping** pe **macro-F1** în validare.

2.3.3 SimpleCNN: efectul augmentărilor și observații din curbele de validare

Am evaluat trei setări:

- **no-aug**: doar normalizare;
- **aug (cu RandomResizedCrop)**: crop+flip+transformări geometrice+color jitter;
- **aug fără crop** (augmentări „safe”): flip + rotații/scale mici + color jitter.

În figura de mai jos sunt comparate curbele pe validare pentru **loss** și **macro-F1**.

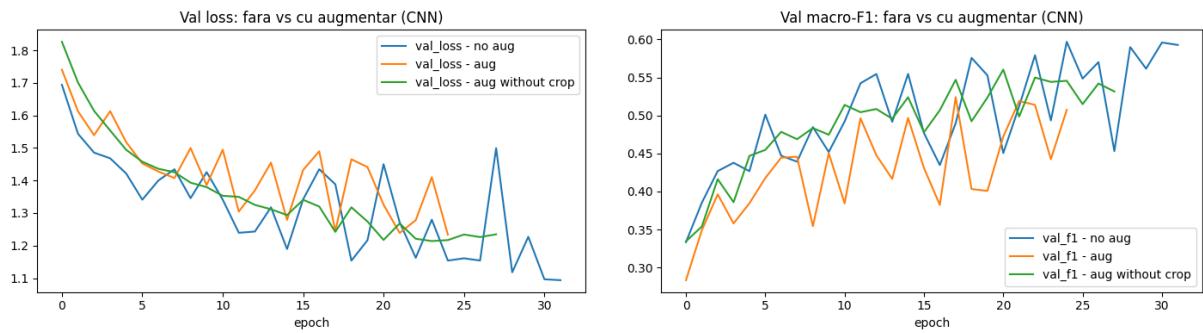


Figura 9: SimpleCNN: comparație pe validare între *no-aug*, *aug* (cu crop) și *aug fără crop*. (Se discută *val_loss* și *val_macro-F1*).

Se observă un rezultat contra-intuitiv: **no-aug** obține cel mai bun macro-F1 pe validare (best $\text{val_F1} \approx 0.597$), în timp ce augmentarea cu crop scade semnificativ performanța (best $\text{val_F1} \approx 0.524$). Explicația este consistentă cu natura datelor: la 96×96 , **RandomResizedCrop** poate elimina obiectul sau poate produce imagini prea „tăiate”, ceea ce introduce etichete cu semnal vizual slab (augmentarea devine zgromot, nu regularizare). Varianta „safe” (fără crop) recuperează parțial (best $\text{val_F1} \approx 0.560$), dar tot rămâne sub no-aug.

În practică, acest rezultat sugerează că pentru acest CNN simplu și pentru acest dataset, augmentările adăugate au fost fie prea perturbatoare, fie nu au fost necesare (setul este deja relativ variat, iar arhitectura are capacitate moderată).

2.3.4 SimpleCNN: tuning pe *learning rate*

Am încercat îmunătățirea variantei **no-aug** prin ajustarea ratei de învățare: $lr=1e-3$ vs $lr=3e-4$. Intuiția este că un *learning rate* mai mic poate produce antrenare mai stabilă și poate reduce oscilațiile în metrii (mai ales când se folosește AdamW).

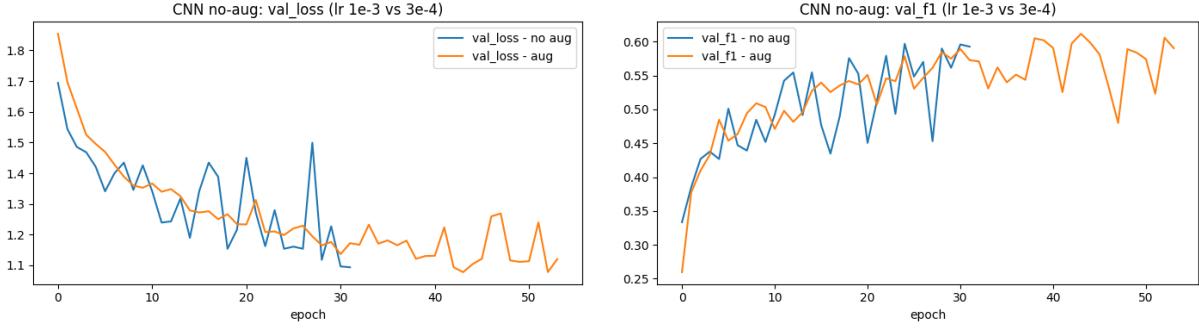


Figura 10: SimpleCNN (no-aug): comparație între $lr=1e-3$ și $lr=3e-4$ (discutăm `val_loss` și `val_macro-F1`).

Rezultatul a fost o îmbunătățire modestă: best `val_F1` ≈ 0.612 pentru $lr=3e-4$, față de ≈ 0.597 la $lr=1e-3$. Totuși, scorul rămâne limitat, sugerând că problema este mai degrabă **capacitatea arhitecturii** decât doar hiperparametrii de optimizare. Acest lucru a motivat schimbarea arhitecturii către un CNN mai expresiv.

2.3.5 Arhitectura CNNv2: diferențe față de SimpleCNN

A doua arhitectură (CNNv2) este mai puternică deoarece:

- folosește **două convoluții per bloc** (*VGG-style*), permitând învățarea unor caracteristici mai complexe înainte de pooling;
- are **mai multă adâncime** (4 blocuri, reducere $96 \rightarrow 48 \rightarrow 24 \rightarrow 12 \rightarrow 6$);
- crește numărul de canale progresiv ($64 \rightarrow 128 \rightarrow 256 \rightarrow 256$), deci capacitate mai mare pentru texturi și forme;
- păstrează regularizarea prin **BatchNorm** și **Dropout + Global Average Pooling**.

Ne așteptam la rezultate mai bune deoarece setul are 10 clase cu variații mari și confuzii între animale, unde sunt necesare caracteristici locale (texturi) și ierarhii de forme, pe care SimpleCNN le poate surprinde insuficient.

2.3.6 CNNv2: comparație no-aug vs aug și interpretarea curbelor spiky

Am repetat experimentul **no-aug** vs **aug (safe)** pentru CNNv2. Pe validare, CNNv2 a obținut:

$$\text{no-aug: best val_F1} \approx 0.7787, \quad \text{aug: best val_F1} \approx 0.7552.$$

Ca și anterior, varianta fără augmentări a ieșit ușor mai bine, iar reducerea lr la $3 \cdot 10^{-4}$ a degradat performanța (best `val_F1` ≈ 0.685), indicând că optimizarea devine prea lentă și modelul nu mai ajunge într-o regiune bună a soluției în numărul de epoci permis.

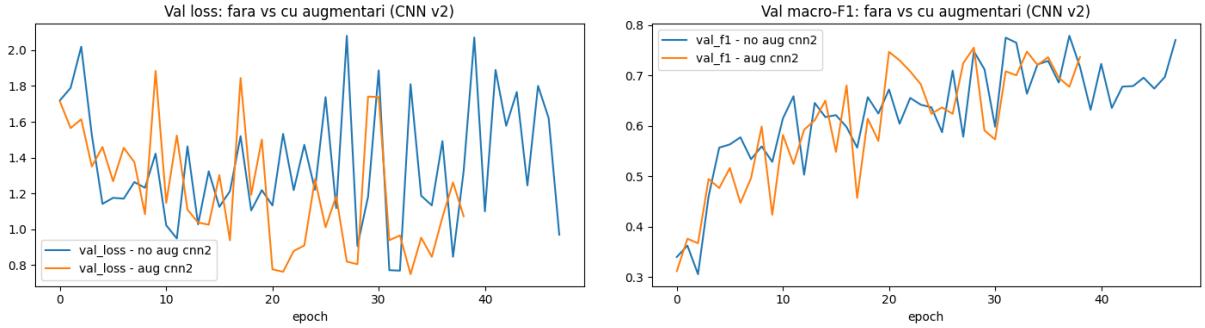


Figura 11: CNNv2: comparație pe validare între *no-aug* și *aug*.

Un aspect notabil este că pentru CNNv2 curba `val_loss` este **foarte „spiky”** (oscilează puternic), deși `val_macro-F1` crește și atinge valori ridicate. Acest lucru nu indică neapărat o problemă, ci reflectă diferența dintre **cross-entropy** și **metricile pe decizie**:

- **Macro-F1** depinde de eticheta prezisă (`argmax`). Dacă modelul clasifică corect aceleasi exemple, F1 poate rămâne mare chiar dacă probabilitățile se schimbă.
- **Loss-ul (cross-entropy)** penalizează puternic cazurile în care modelul este **foarte încrezător, dar greșește**. O singură serie de exemple „dificile” poate produce spike-uri mari în loss, chiar dacă numărul de exemple clasificate corect nu se schimbă dramatic.
- În plus, augmentările stocastice pot produce oscilații. În astfel de situații, **early stopping pe val_F1** este justificat: păstrăm checkpoint-ul cu cea mai bună separare pe clase, chiar dacă loss-ul nu scade monoton.

Prin urmare, am selectat modelul final pe baza **best val_F1**, nu pe forma curbei `val_loss`.

2.3.7 CNNv2: evaluare pe test și interpretarea erorilor

Modelul ales pentru test a fost **CNNv2 no-aug**, $lr=1e-3$, deoarece a avut cel mai bun `val_F1`. Pe test am obținut:

$$\text{loss} = 0.898, \quad \text{accuracy} = 0.7830, \quad \text{macro-F1} = 0.7817.$$

Rezultatele sunt semnificativ mai bune decât MLP ($\text{macro-F1} \approx 0.50$), confirmând că arhitecturile conluviționale învață caracteristici relevante pentru clasificare.

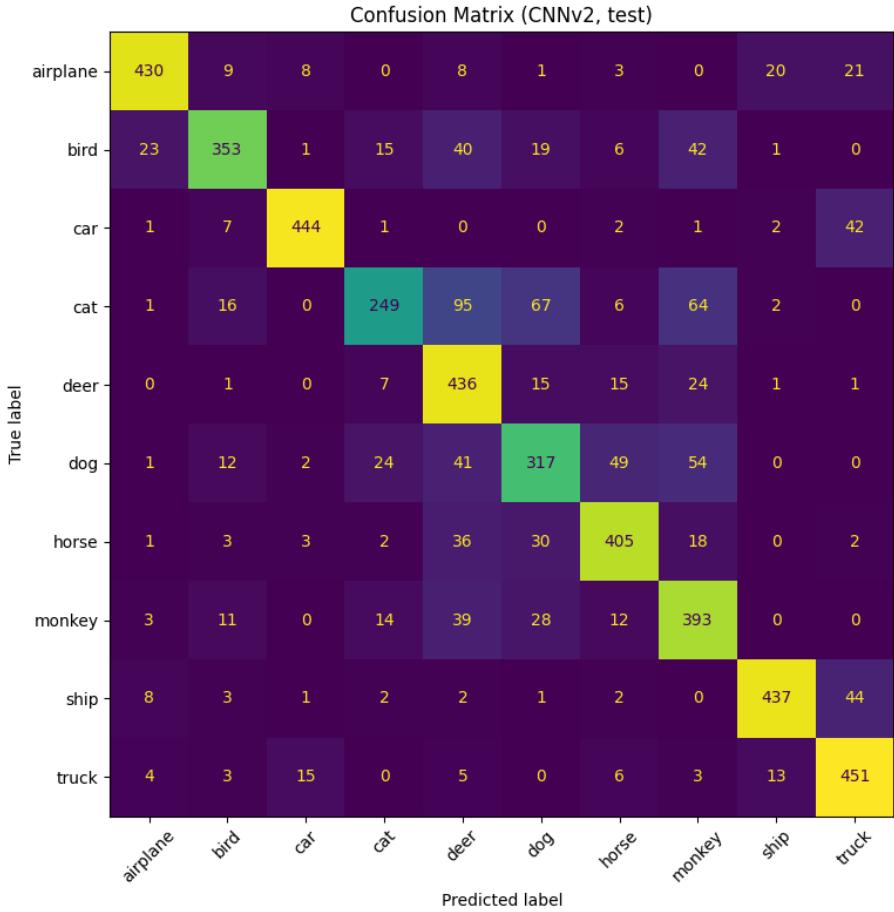


Figura 12: Matricea de confuzie pentru CNNv2 (test).

Matricea de confuzie arată că majoritatea claselor de vehicule sunt recunoscute foarte bine (*car*, *truck*, *ship*, *airplane*), iar principalele confuzii apar între clase de animale cu texturi și forme apropriate (ex. *cat* vs *dog*, *dog* vs *monkey*). Acest tipar este consistent cu EDA: animalele au variație intra-clasă mare și fundaluri similare (natură), iar diferențele pot fi subtile.

Pentru o analiză mai granulară am calculat acuratețea pe clasă (support=500):

Tabela 5: Acuratețe pe clasă pentru CNNv2 (test).

Clasă	acc_per_class	support
cat	0.498	500
dog	0.634	500
bird	0.706	500
monkey	0.786	500
horse	0.810	500
airplane	0.860	500
deer	0.872	500
ship	0.874	500
car	0.888	500
truck	0.902	500

Se observă că *cat* rămâne cea mai dificilă clasă (sub 0.50), iar *truck*/*car* sunt cele mai ușoare (peste 0.88–0.90). Acest lucru sugerează că pentru clasele de animale ar fi utile fie augmentări mai atent calibrate (mai puțin distrugătoare), fie o arhitectură și mai expresivă (sau pre-antrenare/transfer learning), însă pentru scopul temei CNNv2 oferă un compromis foarte bun.

Concluzie CNN. Comparativ cu MLP, CNN-urile oferă un câștig major datorită inductive bias-ului spațial. În cazul *Imagebits*, augmentările agresive de tip crop au degradat performanța (probabil din cauza rezoluției mici și a riscului de a elimina obiectul). Arhitectura CNNv2 (mai adâncă, cu două conv-uri per bloc) a adus cea mai mare îmbunătățire și a obținut performanță solidă pe test.

2.3.8 Reproductibilitate și variația rezultatelor între rulări

Într-o rulare anterioară, cea mai bună configurație pentru CNNv2 no-aug a fost varianta cu $lr=3e-4$, însă în rularea curentă performanța maximă pe validare a fost obținută cu $lr=1e-3$. Această diferență este posibilă deoarece antrenarea nu este complet deterministă: inițializarea aleatorie a greutăților, ordinea mini-batch-urilor (`shuffle`), operațiile paralele/GPU (unele kerneluri pot fi nedeterministe) și chiar fluctuațiile mici ale optimizării pot schimba traectoria în spațiul parametrilor, mai ales când folosim *early stopping* pe un set de validare relativ mic.

Din acest motiv, pentru experimentul de *transfer learning* pe *Land Patches* am păstrat modelul salvat din rularea anterioară ($lr=3e-4$), deoarece la momentul salvării acesta era checkpointul cu cel mai bun scor pe validare, și anume ($valf1=0.80$) și este suficient ca punct de plecare pentru fine-tuning. În secțiunea următoare, modelul preantrenat folosit la fine-tuning va fi deci varianta salvată cu $lr=3e-4$.

3 Analiza setului de date Land Patches

3.1 Descrierea setului de date și analiza exploratorie

3.1.1 Descriere generală și etichete

Setul *Land Patches* conține imagini satelitare (patch-uri) de dimensiune fixă, organizate deja în trei split-uri: `train`, `val` și `test`. Clasele sunt: *{AnnualCrop, Forest, HerbaceousVegetation, Highway, Industrial, Pasture, PermanentCrop, Residential, River, SeaLake}*.

Fiindcă setul include explicit `train/val/test`, nu a fost necesară construirea manuală a unui split de validare (ca la *Imagebits*), ceea ce elimină riscul de *data leakage* dintr-o separare greșită.

3.1.2 Echilibrul claselor și distribuția pe split-uri

Am verificat distribuția claselor pe fiecare split. Datasetul este **perfect echilibrat**: `train`=2000 imagini (200/clasă), `val`=1000 imagini (100/clasă), `test`=7000 imagini (700/clasă). Acest lucru este util deoarece:

- acuratețea devine mai reprezentativă (nu este „umflată” de o clasă dominantă);
- **macro-F1** este o metrică naturală (toate clasele au aceeași importanță).

Tabela 6: Sumar Land Patches: dimensiuni și echilibru pe split.

Split	Total imagini	Imagini/clasă	Număr clase
train	2000	200	10
val	1000	100	10
test	7000	700	10

Figura ?? confirmă vizual această distribuție echilibrată.

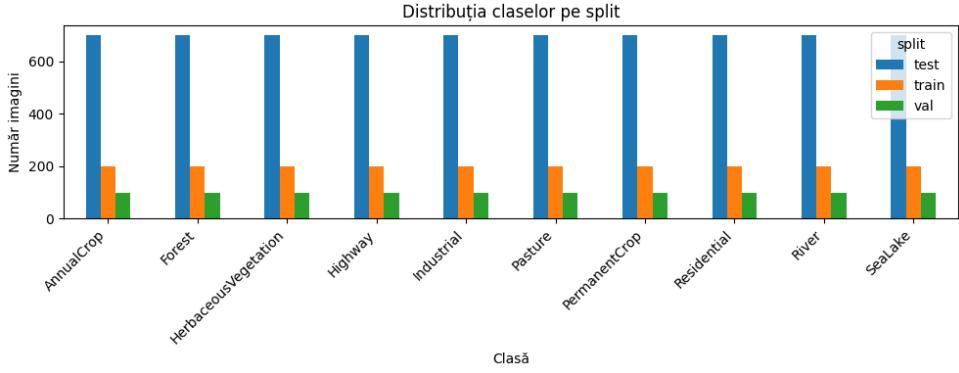


Figura 13: Nr de clase per split

3.1.3 Verificări de integritate (sanity checks)

Am rulat verificări rapide pentru consistența fișierelor:

- **Erori la încărcare:** 0 (pe sample);
- **Dimensiuni unice:** (64, 64) pentru toate imaginile verificate;
- **Mod culoare:** toate imaginile sunt RGB.

Aceste rezultate arată că datele sunt curate și omogene din punct de vedere al formatului, ceea ce simplifică pipeline-ul (aceeași arhitectură și aceeași normalizare pentru toate imaginile).

3.1.4 Exemple vizuale și variabilitate intra/inter-clasă

Am inspectat manual exemple din mai multe clase pentru a evalua:

- **variația intra-clasă** (schimbări de iluminare, anotimp, textură și orientare);
- **separabilitatea inter-clasă** (cât de ușor pot fi distinse clasele doar vizual).

Din exemplele din Fig. 14 se observă:

- Clase precum *Industrial* și *Residential* au **structuri puternic geometrice** (clădiri, rețele), fiind mai „texturate” și cu multe muchii.
- Clase precum *Forest* și *Pasture* au **texturi naturale** și pot varia cromatic (nuanțe de verde), ceea ce crește variația intra-clasă.
- Clasele *River* și *SeaLake* pot fi **vizual similare** (suprafețe mari de apă, tonuri reci), sugerând confuzii posibile între ele fără semnale suplimentare (margini/shoreline, forme).

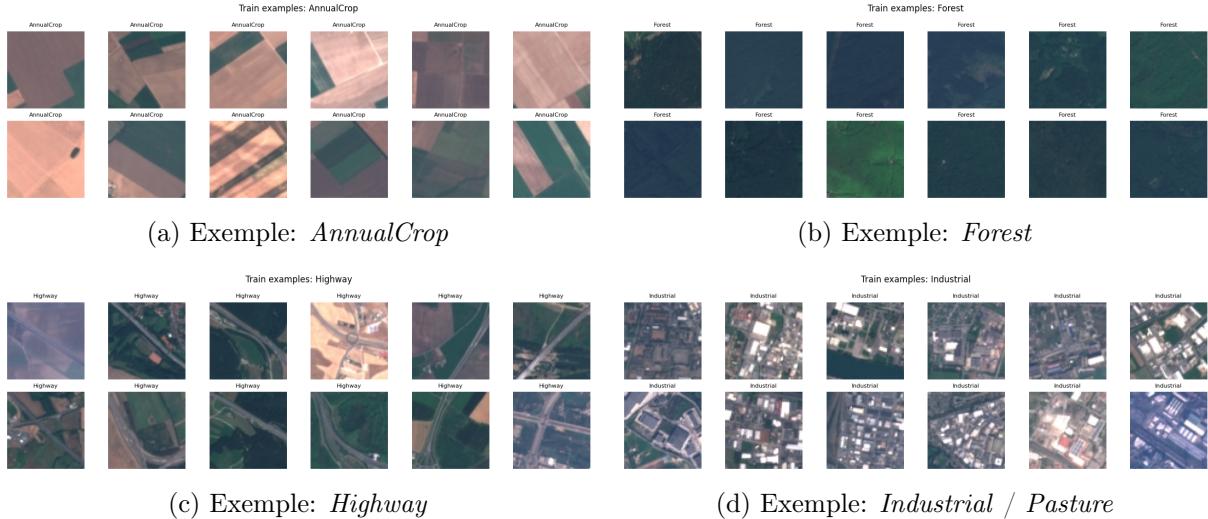


Figura 14: Exemple din Land Patches (train): variabilitate intra-clasă și diferențe inter-clasă.

3.1.5 Statistici fotometrice globale: mean/std pe RGB

Pentru a normaliza imaginile în mod corect înainte de antrenarea rețelelor neurale, am calculat media și deviația standard pe canale RGB (pe **train**) printr-un algoritm de tip *streaming* (evită încărcarea tuturor imaginilor simultan). Rezultatul a fost:

$$\mu = (0.3483, 0.3846, 0.4113), \quad \sigma = (0.2059, 0.1390, 0.1184).$$

Acstea valori sunt folosite ulterior în `Normalize(mean, std)` pentru stabilitate numerică și convergență mai bună.

Tabela 7: Land Patches: statistici globale RGB (train) pentru normalizare.

	R	G	B
mean (μ)	0.3483	0.3846	0.4113
std (σ)	0.2059	0.1390	0.1184

3.1.6 Luminozitate: histogramă (count) vs histogramă (density)

Am analizat distribuția luminozității (*brightness*) pentru **train/val/test**. În Fig. 19 sunt două tipuri de reprezentări:

- **Histogramă cu count:** numărul efectiv de imagini în fiecare bin. Aici **test** apare mult mai „înalt” deoarece are 7000 imagini, față de 2000/1000.
- **Histogramă cu density:** histograma este normalizată astfel încât aria totală să fie 1 pentru fiecare split. Aceasta permite compararea **formei distribuției** independent de mărimea setului.

Se observă că formele distribuțiilor sunt similare (nu există un *domain shift* major de expunere între split-uri), dar apar variații locale (cozi către valori mai luminoase/mai întunecate), ceea ce justifică augmentări moderate de tip *brightness/contrast jitter*.

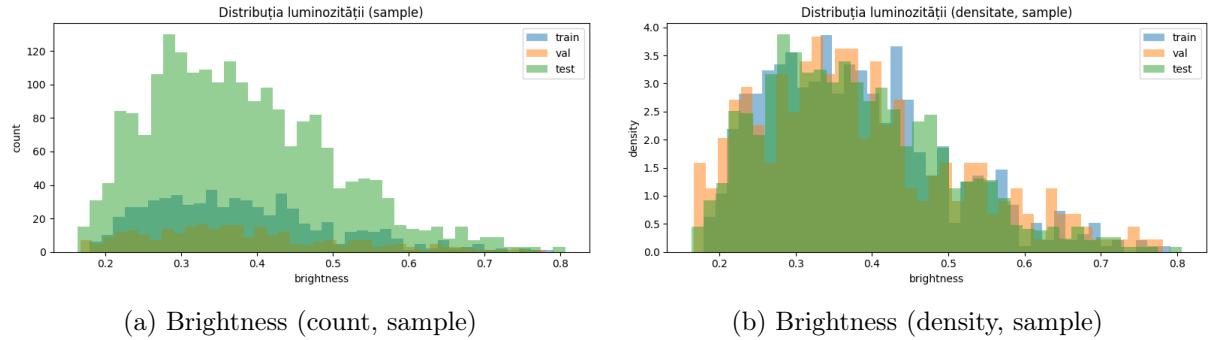


Figura 15: Distribuția luminozității: count vs density (train/val/test).

Pe lângă grafice, am summarizat brightness într-un tabel (mean/std/min/max/median), care confirmă valori apropriate între split-uri:

Tabela 8: Land Patches: statistici brightness pe split (sample).

Split	mean	std	min	max	median
test	0.379	0.119	0.164	0.806	0.363
train	0.381	0.119	0.180	0.792	0.365
val	0.383	0.134	0.167	0.782	0.363

3.1.7 Similaritatea *River* vs *SeaLake*: semnale cromatice

Din inspecția vizuală, *River* și *SeaLake* sunt două clase care pot fi confundate: ambele conțin suprafețe mari de apă și au tonuri reci. Pentru a susține această observație, am comparat distribuții cromatice (R/G/B) între cele două clase (pe sample). Figura 17 sugerează suprapuneri considerabile, deci culoarea singură nu separă perfect clasele; este probabil ca diferențierea să depindă de **textură** și **margini** (shoreline, curbe, structuri).

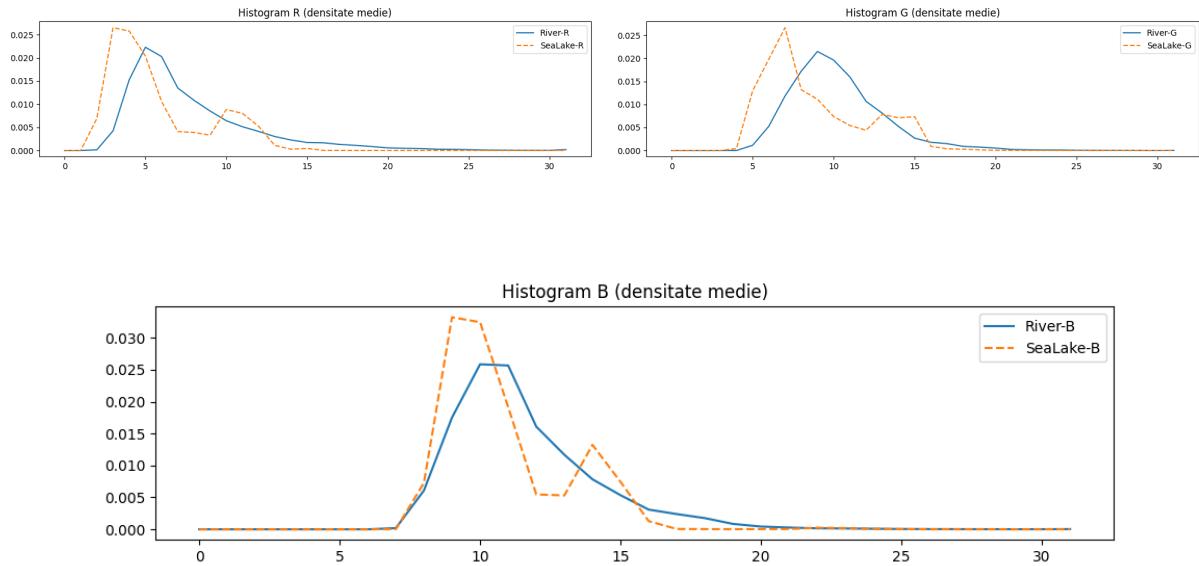


Figura 17: Compararea distribuțiilor RGB (density medie) pentru *River* vs *SeaLake*.

3.1.8 Analiza texturii: *edge strength* (*River* vs *SeaLake*)

Pentru a surprinde diferențe de textură/structură, am introdus o măsură simplă de **edge strength**: media gradientului absolut pe direcțiile *x* și *y* (după conversie la grayscale). Intuiție:

- *River* conține adesea **margini** (maluri, curbe, tranziții apă-uscat) \Rightarrow gradient mai mare;
- *SeaLake* are frecvent suprafețe **omogene** de apă \Rightarrow gradient mai mic.

Figura de mai jos arată exemple cu edge strength **mare** vs **mic** pentru fiecare clasă, confirmând vizual această ipoteză.

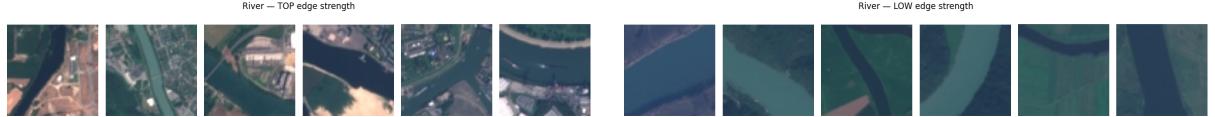


Figura 18: River top vs low edge strength



Figura 19: SeaLake top vs low edge strength

Boxplot-ul de mai jos compară distribuțiile: *River* are în medie edge strength semnificativ mai mare decât *SeaLake*, iar *SeaLake* este concentrat în valori mici (imagini mai uniforme).

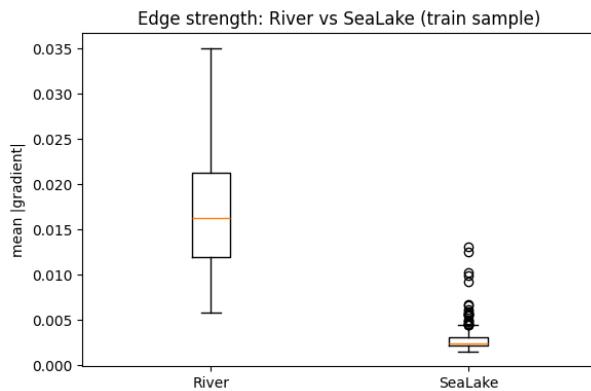


Figura 20: BoxPlot edge strength

3.1.9 Edge density pe toate clasele: interpretare globală

Am calculat edge strength (denumit și *edge density*) și pentru toate clasele (sample), obținând medii distincte. Rezultatele sunt sumarizate în Tabelul 9 și vizualizate în Fig. 21. Se observă că *Industrial* și *Residential* au cele mai mari valori (multe structuri și muchii), în timp ce *SeaLake* are cele mai mici valori (zone omogene).

Tabela 9: Land Patches: statistici edge density pe clasă (train sample).

Clasă	edge_density_mean	edge_density_std
Industrial	0.052876	0.009439
Residential	0.044432	0.015403
PermanentCrop	0.027860	0.010113
Highway	0.023468	0.009364
HerbaceousVegetation	0.021670	0.012822
River	0.017058	0.006208
AnnualCrop	0.013458	0.005433
Pasture	0.010279	0.002974
Forest	0.007366	0.002100
SeaLake	0.002987	0.001636

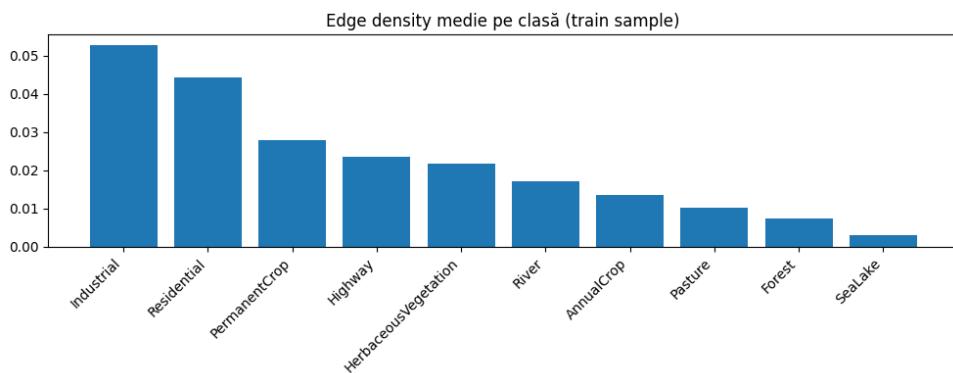


Figura 21: Edge density medie pe clasă (train sample).

Concluzie EDA (Land Patches). Land Patches este echilibrat și coerent ca format (64×64 RGB), fără *domain shift* global evident în luminozitate. Totuși, există clase cu potențial de confuzie (ex. *River* vs *SeaLake*) unde diferențele sunt mai degrabă de **textură** și **margini** decât de culoare. Analiza edge strength confirmă că unele clase sunt mult mai „structurate” (*Industrial/Residential*), în timp ce altele sunt mai omogene (*SeaLake*), ceea ce sugerează că un CNN poate exploata eficient aceste semnale spațiale.

3.2 Model MLP

3.2.1 Motivația augmentărilor (din EDA).

Din analiza exploratorie pentru *LandPatches* am observat:

- **variație intra-clasă** (patch-uri cu texturi similare dar la scări/pozitii diferite în cadrul imaginii);
- **diferențe cromatice moderate** între clase (nuanțe de verde/albastru specifice vegetației vs apă);
- **variații de iluminare/contrast** fără un *domain shift* major între **train/val/test** (distribuții de luminozitate foarte apropiate).

Pentru **no-aug** am aplicat doar normalizarea pe canale folosind media și deviația standard estimate pe **train**. Pentru **aug** am folosit augmentări moderate, motivate de observațiile de mai sus:

- **HorizontalFlip** ($p = 0.5$): în imagini satelitare orientarea stânga-dreapta nu schimbă semnificația semantică; ajută la invarianta la orientare.
- **Shift/Scale/Rotate**: reduce sensibilitatea la mici translații, schimbări de scară și rotații (structuri precum drumuri, parcele sau maluri apar în poziții/orientări diferite).
- **ColorJitter**: modelează variațiile locale de expunere, contrast și saturatie observate în EDA, fără a introduce un *domain shift* artificial.
- **Normalize**: stabilizează optimizarea (intrări la scări comparabile pe canale).

3.2.2 Reprezentarea datelor pentru MLP

Imaginiile au dimensiunea 64×64 și 3 canale RGB. Pentru MLP am folosit o reprezentare *flatten*, adică am transformat fiecare imagine într-un vector 1D:

$$\text{in_dim} = 3 \cdot 64 \cdot 64 = 12288.$$

Această alegere oferă un *baseline* simplu, însă pierde explicit structura spațială (margini, forme, relații locale), ceea ce poate crește confuziile între clase similare vizual (de ex. *Highway* vs *River* sau *River* vs *SeaLake*).

3.2.3 Arhitectura MLP și antrenare

Modelul MLP folosește două straturi ascunse, cu:

- **BatchNorm1d**: stabilizează distribuția activărilor și accelerează convergența;
- **ReLU**: nelinearitate robustă;
- **Dropout**: reduce overfitting-ul, important la input *flatten*.

Am testat două capacitați:

- **small**: $512 \rightarrow 256$ neuroni, dropout mai mare (0.4) pentru regularizare;
- **large**: $1024 \rightarrow 512$ neuroni, dropout 0.3 (capacitate mai mare, risc mai mare de overfitting).

Optimizarea a fost realizată cu AdamW (weight decay = 10^{-4}) și *early stopping* pe **val macro-F1**.

3.2.4 Experimente și selecția modelului

În Tabelul 10 sunt sumarizate cele patru configurații testate (aug/no-aug \times small/large). Criteriul de selecție a fost **val macro-F1** maxim.

Configurație	Best val macro-F1	Observație
no-aug, small	0.5784	converge rapid, apoi oscilează
aug, small	0.6083	generalizare mai bună (best)
no-aug, large	0.5825	capacitate mai mare, overfitting mai ușor
aug, large	0.5971	bun, dar sub aug+small

Tabela 10: Rezultate MLP pe *LandPatches* (criteriu: best **val macro-F1**).

3.2.5 Comparația celor mai bune două configurații

În continuare comparăm cele mai bune două rulari: **(no-aug, large)** vs **(aug, small)**.

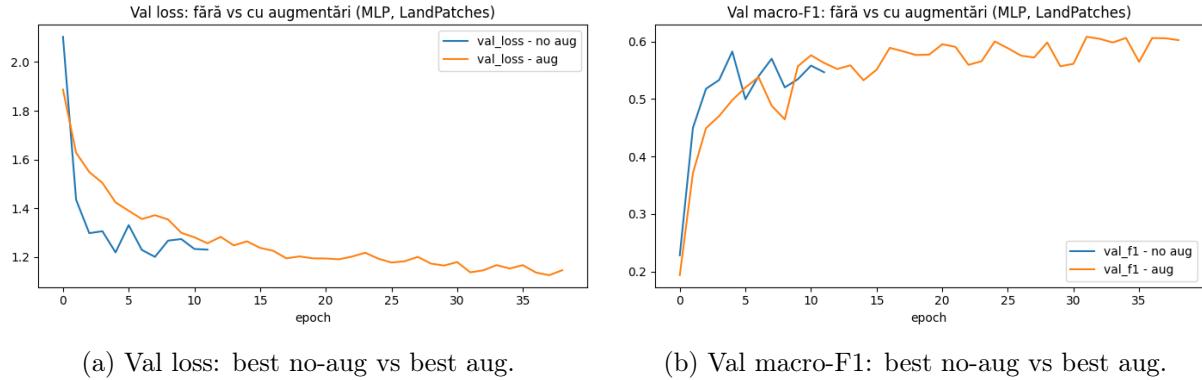


Figura 22: Comparație între cele mai bune două configurații MLP pe *LandPatches*.

Din Figura 22 se observă că varianta cu augmentări (*aug, small*) are o generalizare mai bună:

- **Val loss:** cu augmentări scade mai lent, dar tinde să se stabilizeze la valori mai mici, indicând o potrivire mai robustă (mai puțin dependentă de pattern-uri accidentale din train).
- **Val macro-F1:** crește gradual și depășește configurațiile fără augmentări, ceea ce sugerează un câștig pe clasele mai “grele” (texturi apropiate / separare dificilă).

3.2.6 Evaluare pe test

Modelul ales a fost **MLP (aug, small)** deoarece a obținut cel mai bun **val macro-F1 = 0.6083**. Pe setul **test**, modelul obține:

$$\text{loss} = 1.171, \quad \text{acc} = 0.5914, \quad \text{macro-F1} = 0.5772.$$

Diferența față de **val** este așteptată: **test** este mai mare și include variații suplimentare care penalizează în special clasele cu semnături vizuale similare.

Confusion Matrix (MLP, LandPatches, test)											
True label	AnnualCrop	Forest	HerbaceousVegetation	Highway	Industrial	Pasture	PermanentCrop	Residential	River	SeaLake	
	391	4	26	52	32	38	68	46	32	11	
	0	566	3	0	0	84	0	3	1	43	
	56	10	240	38	27	53	123	126	18	9	
	95	40	30	144	42	99	20	98	129	3	
	14	0	0	40	590	2	2	46	6	0	
	8	9	25	12	1	586	26	26	7	0	
	184	0	40	35	36	56	260	69	18	2	
	27	1	29	31	60	24	19	491	18	0	
	17	82	6	85	18	94	6	25	359	8	
SeaLake	0	73	0	0	5	57	1	40	11	513	

Figura 23: Matricea de confuzie pe **test** pentru MLP (*LandPatches*).

Matricea de confuzie (Figura 23) arată că MLP-ul separă bine clasele cu texturi foarte distinctive (ex.: *Industrial*, *Pasture*, *Forest*), dar are dificultăți la clase care depind de structură spatială (margini/forme continue) pe care *flatten* nu o modelează explicit. În particular:

- **Highway** are acuratețe foarte mică (≈ 0.21), fiind confundată frecvent cu clase ce conțin structuri liniare sau zone mixte (ex.: *Residential*/*River*).
- **HerbaceousVegetation** și **PermanentCrop** sunt dificil de separat (ambele sunt clase “verzi”, cu texturi apropiate).
- **River** vs **SeaLake**: există confuzii naturale deoarece ambele sunt dominate de apă; diferența ține de margini/contur (river are margini mai puternice), informație pe care un MLP o surprinde limitat față de un CNN.

Clasă	Acc. pe clasă	Support
Highway	0.206	700
HerbaceousVegetation	0.343	700
PermanentCrop	0.371	700
River	0.513	700
AnnualCrop	0.559	700
Residential	0.701	700
SeaLake	0.733	700
Forest	0.809	700
Pasture	0.837	700
Industrial	0.843	700

Tabela 11: Performanța MLP pe **test** (*LandPatches*) pe fiecare clasă.

Tabelul 11 confirmă observațiile din matricea de confuzie: clasele cu texturi/culori distinctive (ex.: *Industrial*, *Pasture*, *Forest*) sunt recunoscute mai bine, în timp ce clasele ce necesită structură spațială (ex.: *Highway*) rămân dificile pentru un MLP pe input *flatten*.

3.3 CNN pe *LandPatches* (antrenare de la zero)

3.3.1 Augmentări:

Pentru *imagini satelitare* are sens să adăugăm transformări mai „agresive” geometric:

- **Resize** la dimensiunea cerută de model (64×64 sau 96×96) pentru intrări consistente.
- **Flip orizontal/vertical și rotație cu 90°** — la patch-uri satelitare orientarea absolută nu e relevantă deci aceste augmentări cresc invarianta fără să schimbe clasa.
- **RandomBrightnessContrast** — din EDA am văzut variații moderate de iluminare/contrast între exemple, fără *domain shift* major;
- **Normalize(mean,std)** — folosește statisticile globale estimate prin streaming mean/std.

3.3.2 Arhitectura CNNv2 și semnificația parametrului base.

Am folosit o arhitectură tipică pentru clasificare pe texturi/pattern-uri:

- 4 blocuri convolutionale; fiecare bloc are **două convoluții** 3×3 + **BatchNorm** + **ReLU**, urmate de **MaxPool(2)** pentru a reduce rezoluția spațială și a crește câmpul receptiv.
- **AdaptiveAvgPool(1)** comprimă caracteristicile la un vector global ($C \times 1 \times 1$), apoi un **Linear** produce cele 10 logit-uri.
- base controlează **numărul de canale** în primul bloc: $3 \rightarrow \text{base}$, apoi cresc progresiv ($\text{base}, 2\text{base}, 4\text{base}$). Acesta setează dimensiunea rețelei: **base mai mare** \Rightarrow capacitate mai mare (mai multe filtre), dar și cost computațional mai mare.
- **Dropout** în classifier (≈ 0.3) pentru regularizare.

3.3.3 Setări de antrenare (comune în experimente).

- Optimizator: **AdamW**, $\text{weight_decay} = 10^{-4}$
- Learning rate (în funcție de experiment): $lr = 3 \cdot 10^{-4}$
- **Early stopping** pe **macro-F1** pe **val** (patience 10, $\text{min_delta} 10^{-4}$), deoarece macro-F1 penalizează confuziile între clase chiar și când distribuția e echilibrată.

3.3.4 Experiment 1: 64×64 , base=48, fără vs cu augmentări.

Rezultatele de validare au fost:

- **No-aug:** best val_F1 = 0.8904
- **Aug:** best val_F1 = 0.9147

În grafice se observă că:

- **Val loss** scade rapid în primele epoci și apoi se stabilizează (modelul învăță repede patternuri de textură/structură).
- **Val macro-F1** crește și atinge un platou; cu augmentări platoul e **ușor mai sus** și de obicei curba e **mai stabilă**, semn de generalizare mai bună.

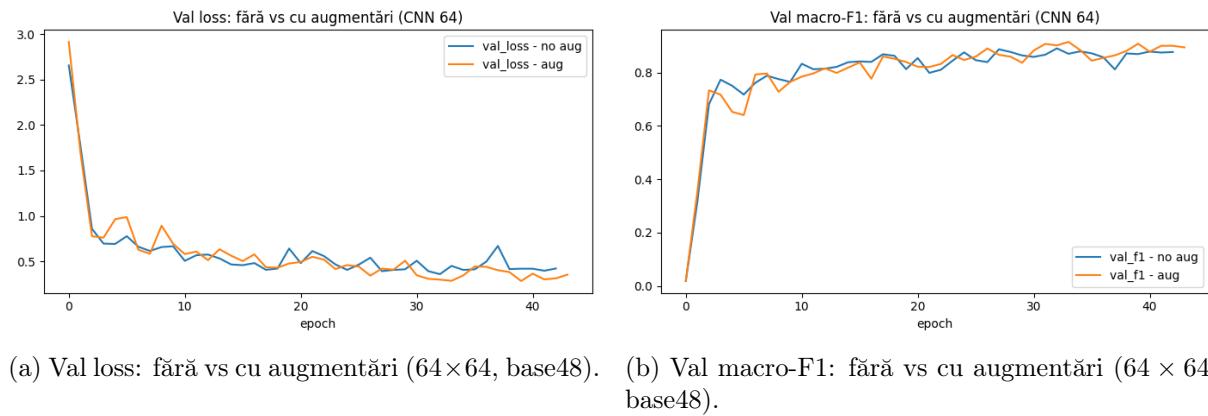


Figura 24: Efectul augmentărilor la 64×64 (CNNv2, base48).

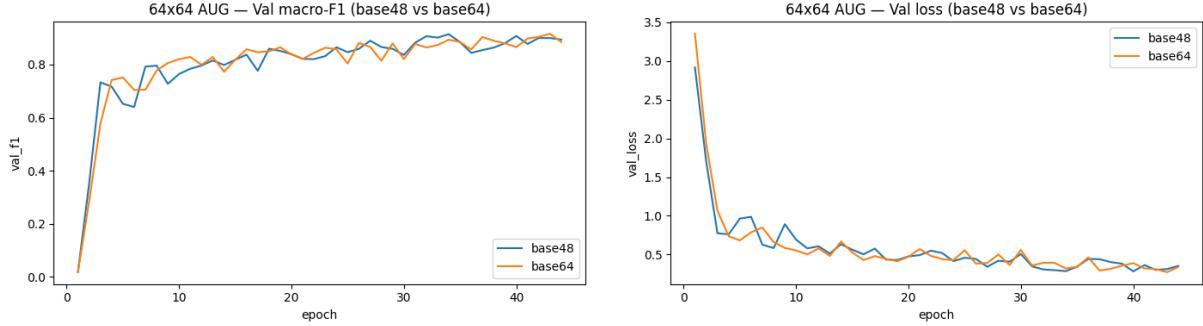
3.3.5 Experiment 2: 64×64 , augmentări, base=48 vs base=64.

Pentru a evalua efectul capacității modelului, am păstrat aceleași augmentări și hiperparametri și am crescut **base**.

- **Scratch 64 base48 (aug):** best val_F1 = 0.9147 (epoch ≈ 34)
- **Scratch 64 base64 (aug):** best val_F1 = 0.9381 (epoch ≈ 63)

Interpretare:

- **base=64** are mai multe filtre, deci poate surprinde mai bine **detalii fine** (texturi agricole, grile urbane, margini de drum/ape).
- Faptul că best-ul apare mai târziu (epoch 63) indică o **învățare mai lentă, dar mai expresivă**; modelul mai mare are nevoie de mai mult timp să-și „așeze” reprezentările fără a supraînvăța.



(a) Val macro-F1: base48 vs base64 (64×64 , aug).

(b) Val loss: base48 vs base64 (64×64 , aug).

Figura 25: Efectul creșterii capacitatei (base) la 64×64 cu augmentări.

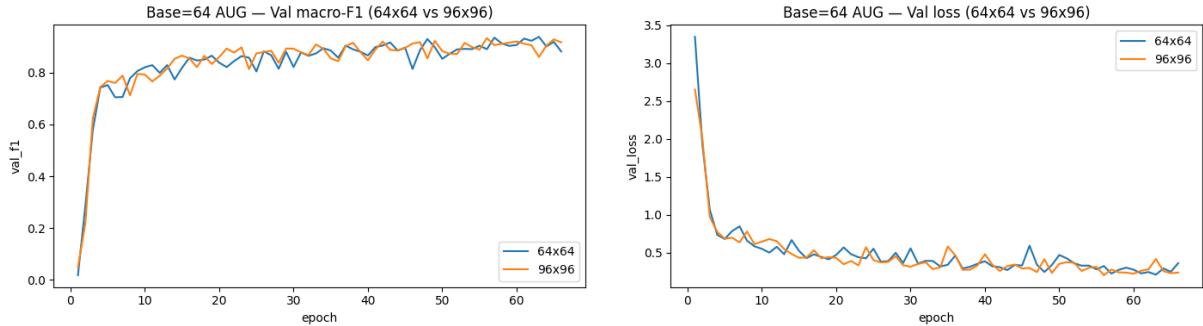
3.3.6 Experiment 3: base=64, augmentări, 64×64 vs 96×96 .

Am antrenat și o versiune 96×96 pentru a avea o bază corectă de comparație cu scenariul de *fine-tuning* (modelul preantrenat pe ImageBits folosește 96×96 și base64).

- **Scratch 64 base64 (aug):** best val_F1 = 0.9381
- **Scratch 96 base64 (aug):** best val_F1 = 0.9333

Observații:

- Diferența este mică, dar 64×64 rămâne **ușor superior**. Un motiv plauzibil: rezoluția originală a datasetului este 64×64 , iar upscaling-ul la 96×96 introduce **interpolare** (poate netezi marginile/texturile), fără să adauge informație reală.
- 96×96 e păstrat în pipeline pentru **comparabilitate** cu fine-tuning, nu neapărat pentru că ar fi optim în regim from-scratch.



(a) Val macro-F1: 64×64 vs 96×96 (base64, aug).

(b) Val loss: 64×64 vs 96×96 (base64, aug).

Figura 26: Efectul schimbării dimensiunii de intrare (base64, augmentări).

3.3.7 Rezumatul experimentelor (validare).

Configurație	Dimensiune	Best val macro-F1
CNNv2 base48, no-aug	64×64	0.8904
CNNv2 base48, aug	64×64	0.9147
CNNv2 base64, aug	64×64	0.9381
CNNv2 base64, aug	96×96	0.9333

Tabela 12: Comparația configurațiilor CNNv2 pe *LandPatches* (setul de validare).

3.3.8 Modelul ales și evaluarea pe test.

Am ales configurația cu cel mai bun scor de validare: **CNNv2, base64, 64×64 , cu augmentări**. Pe setul **test** am obținut:

$$\text{loss} = 0.230, \quad \text{acc} = 0.9257, \quad \text{macro-F1} = 0.9259.$$

Scorul ridicat confirmă că modelul exploatează bine semnăturile structurale specifice claselor (texturi agricole, densitate de muchii urban/industrial, forme de apă/fluviu etc.) și că augmentările alese cresc robusteză fără a distorsiona eticheta.

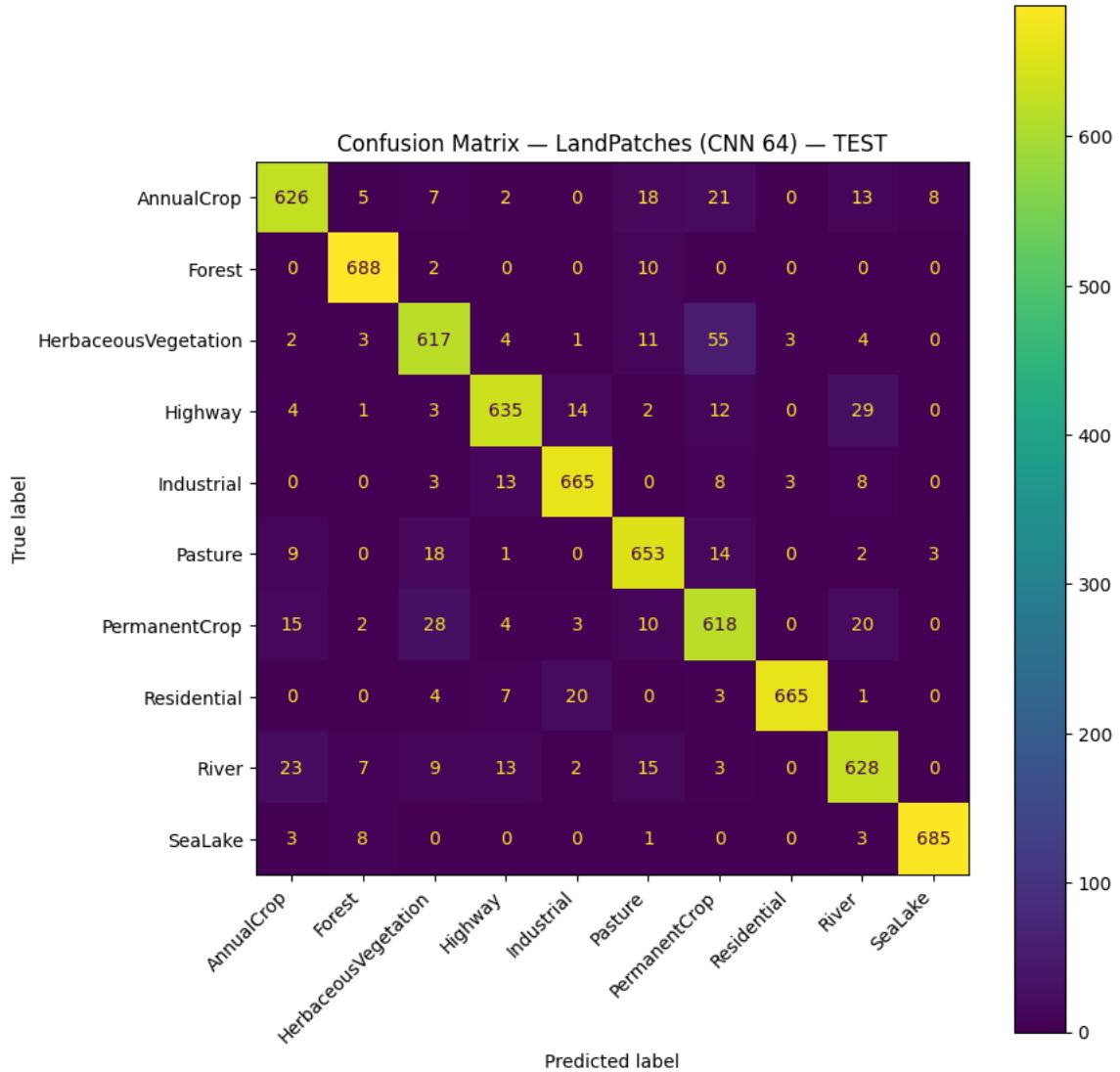


Figura 27: Matricea de confuzie pe **test** pentru CNNv2 (64×64 , base64, aug).

3.3.9 Interpretarea erorilor (din matricea de confuzie și metrii pe clasă).

Matricea este puternic diagonală, ceea ce indică separabilitate bună. Confuzia rămasă este redusă și apare în special între clase **vizual apropiate**:

- *PermanentCrop* vs *HerbaceousVegetation*/*AnnualCrop* — texturi agricole similare, diferențe mai mult de regularitate/parcelare.
- *River* vs *AnnualCrop*/*Pasture* (când apa ocupă puțin din patch sau când malurile sunt dominate de vegetație).

- *SeaLake* are performanță foarte bună (suprafețe uniforme), iar *Forest* rămâne ușor de recunoscut (textură consistentă).

Clasă	Acc / clasă	F1 / clasă	Support
PermanentCrop	0.882857	0.861925	700
HerbaceousVegetation	0.881429	0.887132	700
River	0.897143	0.892045	700
AnnualCrop	0.894286	0.905933	700
Pasture	0.932857	0.919718	700
Highway	0.907143	0.920957	700
Industrial	0.950000	0.946619	700
Residential	0.950000	0.970095	700
Forest	0.982857	0.973126	700
SeaLake	0.978571	0.981375	700

Tabela 13: Performanță pe clasă pe **test** pentru CNNv2 (64×64 , base64, aug).

3.3.10 Motivatie antrenare varianta 96×96 (deși 64×64 e mai bună).

Scopul antrenării la 96×96 a fost să avem o **bază comparabilă** cu etapa de *fine-tuning* (model preantrenat pe *ImageBits* folosește 96×96 și base64). Astfel, în secțiunea de fine-tuning putem separa efectul preantrenării de efectul dimensiunii de intrare/architecturii.

3.4 Fine-tuning CNN pe *LandPatches* folosind un checkpoint preantrenat pe *ImageBits*

Deși *ImageBits* (obiecte naturale) și *LandPatches* (imagini satelitare) sunt domenii diferite, un backbone CNN preantrenat poate transfera **feature-uri generale** utile:

- margini, colțuri, texturi, contraste sunt **universale** pentru imagini RGB;
- în *LandPatches*, diferențierea claselor depinde puternic de **texturi și structuri** (ex.: zone industriale vs rezidențiale, tipare de vegetație), exact zona în care un backbone preantrenat ajută la convergență și generalizare;
- checkpoint-ul ales din *ImageBits* avea deja performanță bună, deci oferă un punct de plecare stabil (în loc să învățăm filtrele de la zero).

3.4.1 Compatibilizare: rezoluție și augmentări (96×96)

Pentru fine-tuning am folosit **aceeași rezoluție ca în preantrenare** (96×96) și aceeași arhitectură CNNv2 cu **base=64**, astfel încât greutățile backbone-ului să fie compatibile dimensional. Augmentările sunt aceleași ca în varianta CNN-from-scratch la 96×96 (flip-uri, rotații, brightness/contrast), potrivite pentru scene satelitare:

- **HorizontalFlip/VerticalFlip + RandomRotate90**: orientarea scenei nu este relevantă semantic;
- **RandomBrightnessContrast**: modelează variații moderate de iluminare/contrast observate în EDA;
- **Normalize**: în cod s-a păstrat normalizarea cu statisticile *LandPatches* (`mean_rgb/std_rgb`) pentru a evita un mismatch de distribuție (deși checkpoint-ul conține și `mean_ib/std_ib` pentru referință).

3.4.2 Încărcarea backbone-ului preantrenat

Checkpoint-ul din *ImageBits* conține `model_state_dict`. Pentru transfer, am încărcat **doar backbone-ul** (straturile `features`), excludând head-ul. Mesajul de încărcare confirmă comportamentul așteptat:

- **Missing keys** sunt doar în `classifier` (normal: head nou, neinitializat din checkpoint);
- **Unexpected keys** este gol (compatibilitate corectă a backbone-ului).

3.4.3 Strategie de fine-tuning în două etape (two-stage)

Pentru a stabiliza adaptarea:

1. Stage 1: freeze backbone (5 epoci)

Se antrenează doar head-ul (`classifier`) cu LR mai mare (`head_lr=1e-3`). Scopul este ca head-ul să învețe o separare inițială pentru noile clase folosind feature-uri deja bune.

2. Stage 2: unfreeze + fine-tune

Se deblochează backbone-ul și se folosește **learning rate diferențiat**:

- backbone: `ft_backbone_lr=1e-4` (pași mici, ajustări fine),
- head: `ft_head_lr=5e-4` (adaptare mai rapidă la clase).

Se aplică **early stopping** (patience 10) pe *val macro-F1*.

Observații din log-uri. În Stage 1, *val macro-F1* crește gradual (până la ~ 0.51), ceea ce e normal deoarece backbone-ul e înghețat și head-ul se aliniază la noile etichete. Imediat după Stage 2, *val macro-F1* crește puternic (ex.: ~ 0.71 din prima epocă), indicând că **ajustarea fină a filtrelor** backbone-ului este critică pentru domeniul satelitar. Valoarea maximă pe validare a ajuns la ***best val macro-F1*** ≈ 0.936 , după care early stopping a oprit antrenarea când nu a mai existat progres susținut.

3.4.4 Evaluare pe test și comparație corectă cu varianta *from scratch*

Pentru o comparație echitabilă, am evaluat:

- **Fine-tune (96, base=64):** acc=0.9279, macro-F1=0.9279, loss=0.223
- **Scratch (96, base=64):** acc=0.9200, macro-F1=0.9195, loss=0.239

Fine-tuning-ul aduce un câștig de aproximativ **+0.008** în *macro-F1* și reduce loss-ul, deci generalizează mai bine.

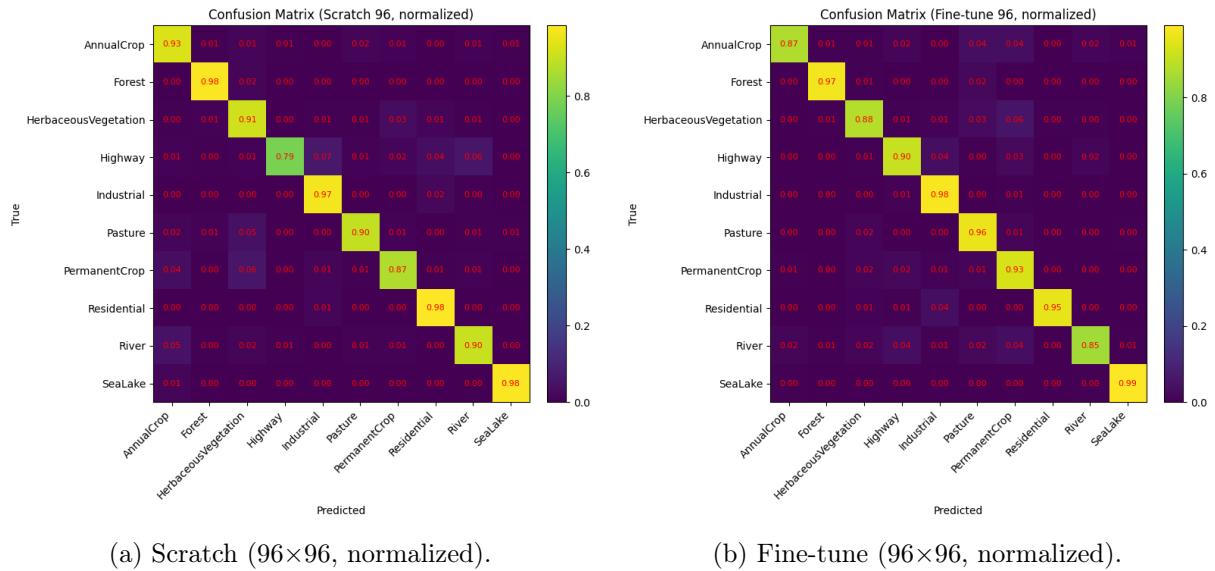
Model	Test loss	Test acc	Test macro-F1
Scratch (96×96, base=64)	0.239	0.9200	0.9195
Fine-tune (96×96, base=64)	0.223	0.9279	0.9279

Tabela 14: Comparație pe test între antrenare de la zero și fine-tuning (setări identice de rezoluție și `base`).

3.4.5 Analiză pe clase: per-class F1 și matrici de confuzie

Tabelele per-clasă arată că fine-tuning-ul îmbunătățește în special clasele mai „grele” structurale:

- **Highway**: F1 crește semnificativ (de la ~ 0.87 la ~ 0.90), ceea ce sugerează că backbone-ul preantrenat oferă feature-uri mai bune pentru tipare liniare/benzi;
- **Residential și HerbaceousVegetation**: câștiguri moderate, consistente (texturi repetitive / structuri fine);
- clase deja foarte ușoare (ex.: **SeaLake, Forest**) rămân aproape saturate, deci diferența e mică.



Chiar dacă domeniile sunt diferite, CNN-urile învață în straturile timpurii **primitve vizuale generale** (margini, contraste, texturi), care sunt utile și în satelit (delimitări de câmpuri, drumuri, clădiri, variații de vegetație). Fine-tuning-ul ajustează apoi aceste filtre către statisticele specifice *LandPatches*, iar strategia în două etape (înghețare apoi deblocare cu LR mic pe backbone) oferă un compromis bun între:

- păstrarea reprezentărilor utile din preantrenare,
- adaptarea lor la noul domeniu fără supra-ajustare.

Concluzie. Modelul **fine-tuned** este cel mai bun dintre variantele comparate la setări identice (96×96 , base=64), obținând **macro-F1=0.9279** pe test și îmbunătățind mai ales clasele mai dificile din punct de vedere textural/structural.

4 Concluzii generale

Pentru *ImageBits*, analiza exploratorie a evidențiat variație intra-clasă ridicată și un bias cromatic (ex. fundal albastru la *airplane/ship*). Rezultatele experimentale confirmă că un *MLP* aplicat direct pe pixeli tinde să facă overfit și nu generalizează la fel de bine, în timp ce un *CNN* exploatează structura spațială a imaginilor și obține performanțe mai bune.

Pentru *LandPatches*, separarea claselor este dominată de texturi și tipare structurale (zone construite, drumuri, modele agricole), iar un *CNN* antrenat de la zero a atins performanțe ridicate. În plus, *fine-tuning*-ul pornind de la un model preantrenat pe *ImageBits* a adus un câștig mic, dar consistent.