Deepfake Classification

Ana-Maria Rusu - Grupa 242

Iunie 2025

1 Introducere

Toate experimentele au fost realizate in mediul Kaggle folosind GPU-uri Tesla T4, iar codul a fost scris si rulat direct in notebook-urile Kaggle.

Au fost explorate doua directii principale: modelul KNN, utilizat pentru evaluarea performantelor pe seturi de caracteristici simple si modelul CNN. Pentru fiecare model s-au testat multiple variante, modificand parametri precum numarul de bins, distanta, tehnici de regularizare sau augmentare (pentru CNN).

2 Modelul KNN

2.1 Idee

Modelul KNN a fost utilizat ca o prima solutie de referinta, pentru a evalua potentialul histogramelor in clasificarea imaginilor deepfake. Alegerea s-a bazat pe simplitatea algoritmului si pe faptul ca nu necesita un proces foarte complex de antrenare. Desi acuratetea obtinuta nu a fost una ridicata, experimentul a permis observarea modului in care variatii precum numarul de bins, distanta folosita si aplicarea unui filtru de netezire influenteaza performanta. Astfel, KNN a servit ca baza de comparatie pentru testarea ulterioara a unor modele mai avansate.

2.2 Proces de implementare

Prima etapa a constat in importarea librariilor necesare pentru prelucrarea imaginilor, manipularea datelor, vizualizarea rezultatelor si implementarea modelului KNN. Librariile utilizate au fost:

- os si cv (OpenCV) pentru citirea si procesarea imaginilor
- numpy si pandas pentru manipularea datelor
- matplotlib.pyplot pentru afisarea graficelor si a matricii de confuzie

- module din scikit-learn pentru scalarea datelor, antrenarea modelului KNN si evaluare

Urmatoarea etapa a implicat procesarea imaginilor si transformarea lor intr-un format numeric adecvat. Am definit o functie care primeste ca parametri:

- cvsPath – calea catre fisierul .csv ce contine numele imaginilor si etichetele (daca acestea exista)
- imaginePath – folderul in care se afla imaginile
- dimensiune – dimensiunea imaginilor dupa redimensionare (implicit 100x100 pixeli)
- areEtichete – un flag boolean care indica daca avem sau nu etichete (pentru setul de test nu avem)

Fiecare imagine este procesata astfel: este citita, dupa care este redimensionata la 100x100 pixeli pentru a avea uniformitate. In continuare, este convertita la format float32 si normalizata prin impartirea la 255, asigurandu-se faptul ca valorile pixelilor sunt in intervalul $[0, 1]$;

Reprezentarea cu histograme:

Pentru fiecare imagine procesata, se concep 3 histograme – cate una pentru fiecare canal de culoare: Red, Green, Blue. Cele 3 canale sunt etichetate in felul urmator: 0 = Blue, 1 = Green, 2 = Red. Histogramele sunt construite pe baza valorilor pixelilor din imagine, care au fost anterior normalizate. Intervalul $[0, 1]$ este impartit in mai multe intervale egale (numite bins), iar pentru fiecare bin se numara cati pixeli au valori care se incadreaza in acel interval. Astfel, histograma reflecta distributia pixelilor pe valorile de culoare dintr-un anumit canal.

Pentru fiecare canal se calculeaza o histograma folosind un numar fix de bin-uri. Histograma rezultata este normalizata astfel incat suma valorilor sale sa fie 1 – aceasta operatie este importanta deoarece astfel se elimina influenta dimensiunii absolute a imaginii asupra modelului. Dupa normalizare, histograma este transformata intr-un vector unidimensional (folosind metoda flatten) si adaugata la vectorul final de caracteristici.

In final, cele 3 histograme sunt concatenate intr-un singur vector de caracteristici cu lungimea $3\times$ numarul de bin-uri. Acest vector va reprezenta imaginea in modelul de clasificare. Functia astfel definita returneaza un tuplu: un array numpy care contine toti vectorii de caracteristici obtinuti pentru setul de imagini si, daca este cazul (pentru datele de antrenare si validare), etichetele corespunzatoare fiecarei imagini. Daca imaginile nu sunt etichetate (cum este in cazul setului de test), al doilea element din tuplu va contine doar id-urile imaginilor.

Definirea si antrenarea modelului:

In etapa de antrenare a modelului, s-a utilizat un clasificator de tip K-Nearest Neighbors (KNN), care clasifica imaginile in functie de cei mai apropiati k vecini din setul de date de antrenament.

Pentru selectarea valorii parametrului k (numarul de vecini) am ales urmatoarea lista :$\{1, 3, 5, 9, 13, 21, 35, 55, 89, 145\}$. Aceste valori au fost alese pe baza sirului lui Fibonacci, impunand o restrictie suplimentara: in cazul in care o valoare este para, aceasta a fost inlocuita cu urmatorul numar impar imediat superior. Scopul acestei reguli este de a evita situatiile de egalitate in procesul de "votare" al vecinilor, ceea ce poate duce la ambiguitati in clasificare. In acest fel, lista acopera o gama destul de larga de scenarii.

Valoarea maxima din lista, k = 145, a fost aleasa pentru a observa comportamentul modelului si in cazul unui numar mai mare de vecini. In diverse surse se sugereaza ca o valoare potrivita pentru k poate fi estimata prin $\sqrt{n}$, unde n reprezinta numarul total de exemple din setul de antrenare. In cazul nostru, avand aproximativ 12500 de imagini in setul de antrenament, aceasta estimare ar indica o valoare apropiata de $k \approx 112$.

Astfel, pentru fiecare valoare k din multimea mentionata, s-a construit si evaluat un model de clasificare. Fiecare model a fost implementat sub forma unui pipeline compus din doua etape/parti:

- StandardScaler – etapa de preprocesare numerica cu rolul de a scala vectorii de caracteristici, astfel incat fiecare dimensiune (caracteristica) sa aiba medie zero si deviatie standard unu.
- KNeighborsClassifier – clasificatorul propriu-zis.

Clasificatorul a fost configurat cu urmatorii parametrii:

- n_neighbors = k – numarul de vecini considerati pentru clasificare
- weights = 'distance' – fiecare vecin contribuie la decizia finala proportional cu distanta fata de entitatea testata. Cu cat este mai apropiat, cu atat are o influenta mai mare;
- metric = 'euclidean' sau 'manhattan' – s-au testat ambele variante pentru compararea performantelor in functie de tipul de distanta utilizata. Detaliile pentru aceasta parte vor fi analizate in urmatoarea sectiune.

Pentru fiecare valoare a lui k, s-a inregistrat acuratetea obtinuta pe setul de validare. Dupa ce s-au parcurs toate valorile posibile, s-a identificat valoarea optima k – aceea care a dus la cea mai buna acuratete. Modelul asociat a fost considerat cel final, iar pe baza acestuia s-au refacut predictiile pe setul de

validare pentru a putea construi matricea de confuzie si a analiza performanta detaliata pe fiecare clasa in parte.

2.3 Interpretarea rezultatelor

Dupa cum a fost mentionat in sectiunea anterioara, experimentul a fost realizat utilizand un numar fix de bins si doua tipuri de distante. In continuare, vor fi analizate modelele testate, cu interpretarea rezultatelor pe baza graficelor si a matricelor de confuzie, urmate de o comparatie intre configuratii.

Distanta euclidiana
Pentru acest caz, au fost testate configuratii cu 64, 32 si 16 bins. S-a observat ca reducerea numarului de bins are, in general, un efect pozitiv asupra acuratetii modelului.

Pentru varianta cu 64 bins am obtinut o acuratete de 0.5560 pentru valoarea lui k = 9.

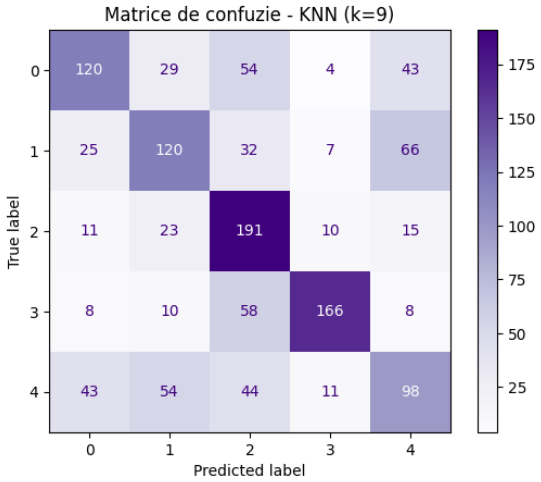

Figure 1: Matricea de confuzie pentru k = 9

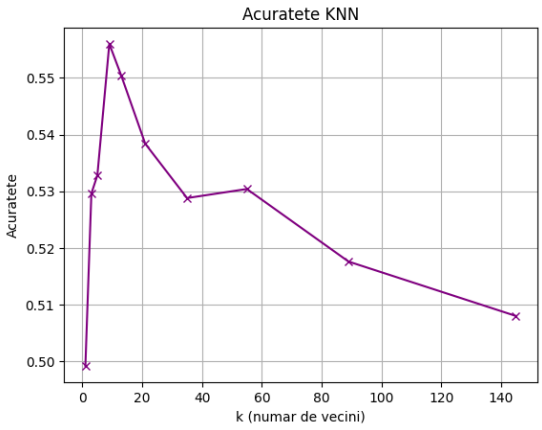

Figure 2: Acuratetea in functie de k (KNN)

Matricea de confuzie pentru modelul KNN (k = 9) evidentiaza faptul ca cea mai bine clasificata categorie este clasa 2, cu 191 de clasificari corecte. In schimb, clasa 4 sufera cele mai mari probleme.

Pentru varianta cu 32 bins am obtinut o acuratete de 0.5568 pentru valoarea lui k = 21.

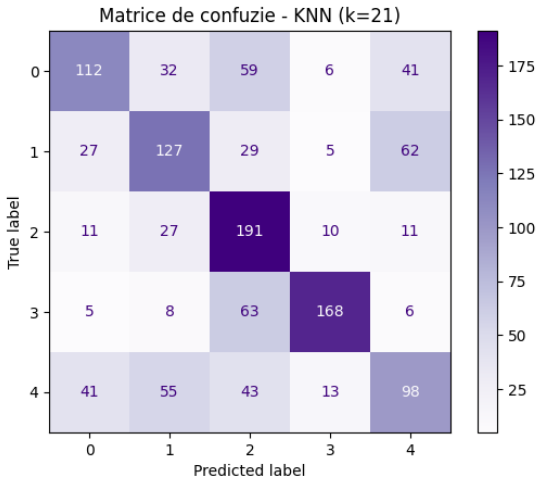

Figure 3: Matricea de confuzie pentru k = 21

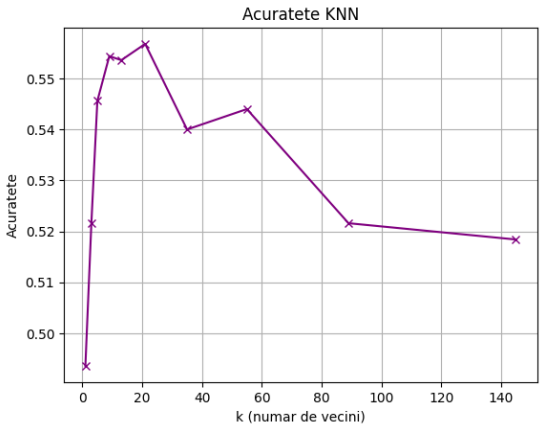

Figure 4: Acuratetea in functie de k (KNN)

Pentru valoarea k = 21, matricea de confuzie arata o distributie similara cu cea pentru k = 9 si 64 bin, insa cu mici scaderi ale numarului de clasificari corecte pentru clasele 0 si 3. Cea mai bine prezisa ramane clasa 2 (191 exemple corecte), in timp ce clasa 4 continua sa fie sursa principala de confuzie, cu multe exemple clasificate gresit ca 0, 1 sau 2.

Pentru varianta cu 16 bins am obtinut o acuratete de 0.5824 pentru valoarea lui k = 13.

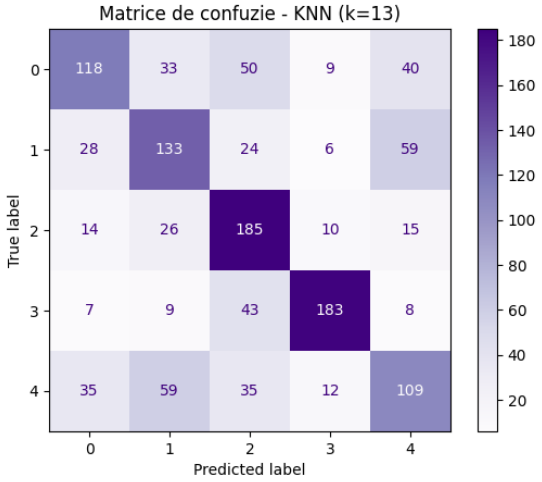

Figure 5: Matricea de confuzie pentru k = 13

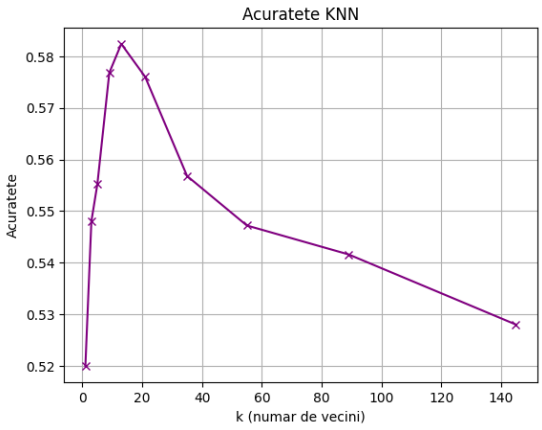

Figure 6: Acuratetea in functie de k (KNN)

Pentru varianta cu 16 bins, modelul KNN cu k = 13 a generat o matrice de confuzie, prin comparatie, mai echilibrata, cu un numar mai ridicat de clasificari corecte pentru clasele 2 (185), 3 (183) si 4 (109). Comparativ cu variantele precedente, modelul reduce confuziile pentru clasa 4 si mentine o precizie ralativ buna pentru celelalte clase.

Distanta manhattan
Pentru acest caz, au fost testate aceleasi configuratii ca pentru distanta euclidiana.

Pentru varianta cu 64 bins am obtinut o acuratete de 0.5540 pentru valoarea lui k = 9. O diferenta de 0.0020 comparativ cu distanta euclidiana.

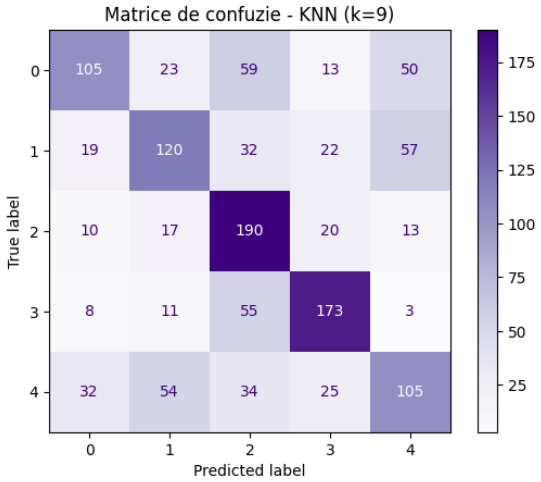

Figure 7: Matricea de confuzie pentru k = 9

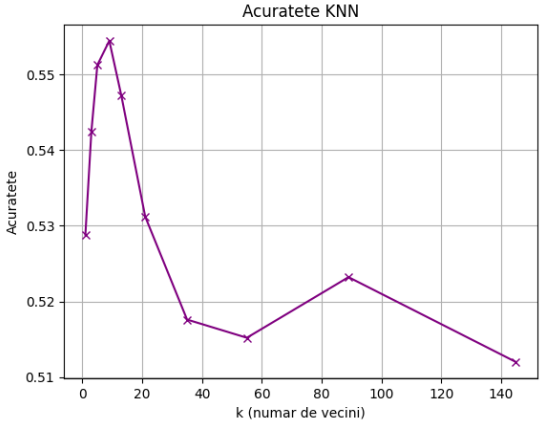

Figure 8: Acuratetea in functie de k (KNN)

In varianta cu 64 bins modelul KNN cu k = 9 obtine o performanta comparabila cu varianta euclidiana, cu 190 de clasificari corecte pentru clasa 2 si 105 pentru clasa 4.

Pentru varianta cu 32 bins am obtinut o acuratete de 0.5704 pentru valoarea lui k = 13.

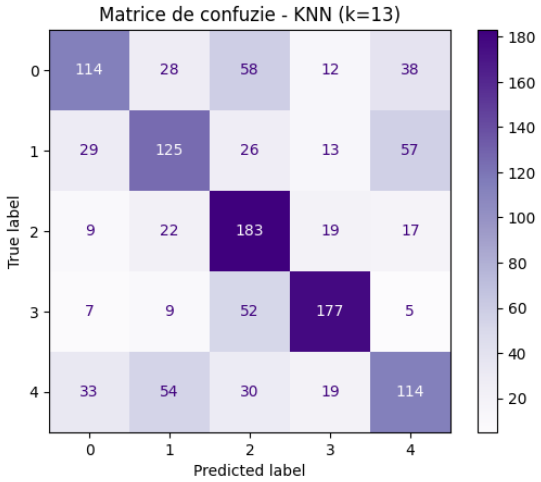

Figure 9: Matricea de confuzie pentru k = 13

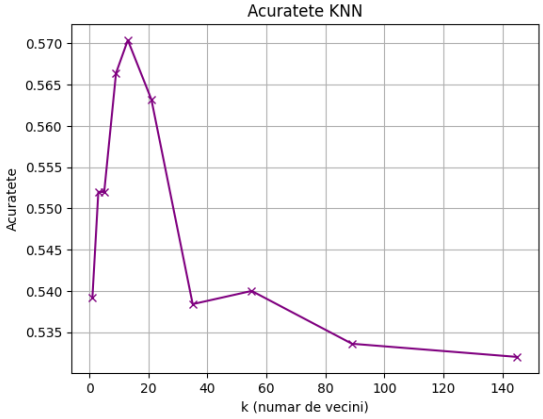

Figure 10: Acuratetea in functie de k (KNN)

Pentru varianta cu 32 bins modelul obtine o distributie destul de echilibrata a clasificarii. Clasa 2 (183 exemple corecte) si clasa 3 (177) sunt in continuare recunoscute bine, iar clasa 4 prezinta o imbunatatire (114 clasificari corecte) fata de alte configuratii.

Pentru varianta cu 16 bins am obtinut o acuratete de 0.5864 pentru valoarea lui k = 13.

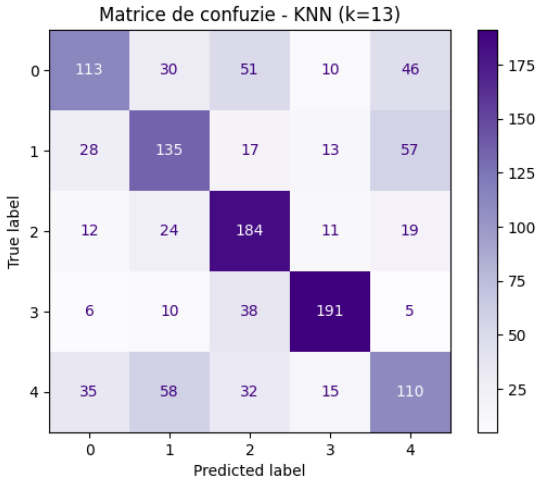

Figure 11: Matricea de confuzie pentru k = 13

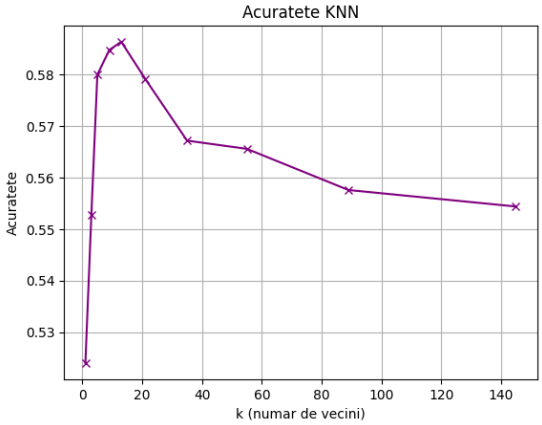

Figure 12: Acuratetea in functie de k (KNN)

In varianta cu 16 bins modelul ofera rezultate echilibrate. Clasa 3 este clasificata foarte bine (191 exemple), iar clasa 4 (110) are o performanta mai buna fata de modelele cu distanta Euclidiana, iar clasa 1 este bine prezisa (135).

2.4 Optimizari

In continuarea procesului de implementare, am selectat cele mai bune doua variante si am aplicat un filtru Gaussian pentru reducerea zgomotului si a detaliilor

din imagini. Acest filtru realizeaza o netezire pe o zona de 5x5 pixeli in jurul fiecarui punct din imagine.

Modelele alese au fost cele care au obtinut cea mai mare acuratete pentru fiecare dintre cele doua tipuri de distante. Astfel, am ales sa incerc imbunatatirea performantelor pentru varianta cu 16 bins, atat pentru distanta Euclidiana, cat si pentru distanta Manhattan.

In urma aplicarii filtrului, pentru 16 bins si distanta Euclidiana (k = 35), am obtinut o acuratete de 0.5768, usor mai slaba decat in varianta fara filtrare. In schimb, pentru aceeasi configuratie cu distanta Manhattan, acuratetea a crescut la 0.5904.

2.5 Concluzie generala asupra modelelor KNN

Rezultatele pentru modelul KNN descris sunt rezumate in tabelul de mai jos:

	bins=16 / k	bins=32/ k	bins=64/ k
distanta euclidiana	0.5824 / 13	0.5568 / 21	0.5560 / 9
distanta manhattan	0.5864 / 13	0.5704 / 13	0.5540 / 9
cu filtru Gaussian	bins=16 / k		
distanta euclidiana	0.5768/35		
distanta manhattan	0.5904/35		

Figure 13: Rezumat acuratete

Pe baza experimentelor realizate, modelul KNN s-a dovedit a fi un clasificator stabil si interpretabil. Acuratetea a fost influentata atat de numarul de bin-uri, cat si de tipul distantei folosite si de preprocesarea aplicata.

Cea mai buna performanta a fost obtinuta pentru configuratia cu 16 bins, distanta Manhattan si k = 13, unde s-a atins o acuratete de 0.5864. Aplicarea filtrului Gaussian a imbunatatit in continuare performanta pentru aceasta varianta, ducand la o acuratete maxima de 0.5904 (k = 35). Acest rezultat confirma faptul ca reducerea zgomotului din imagine poate sprijini procesul de clasificare atunci cand se foloseste distanta Manhattan.

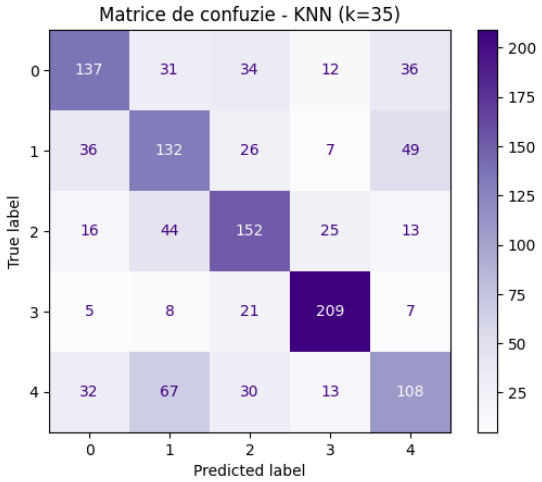

Figure 14: Matricea de confuzie pentru k = 35 si distanta Manhattan

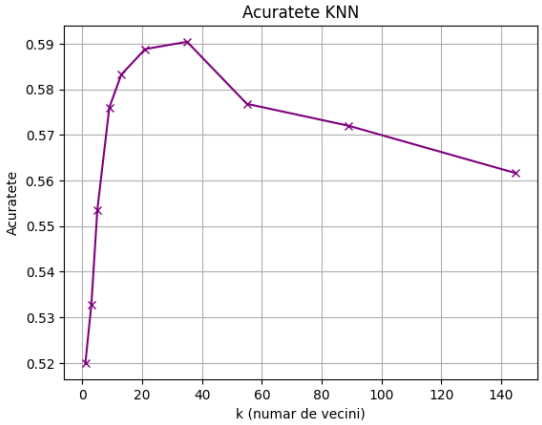

Figure 15: Acuratetea in functie de k (KNN) si distanta Manhattan

Desi modelul KNN a reusit sa depaseasca baseline-ul de aproximativ 0.37466, acuratetea obtinuta nu a fost suficient de buna pentru a justifica incarcarea fisierului CSV in competitia Kaggle. Din acest motiv, am decis sa continui cu un alt model, bazat pe o arhitectura diferita.

3 Modelul CNN

3.1 Idee

Modelul CNN a fost ales ca o solutie mai avansata, capabila sa invete automat caracteristici vizuale relevante din imagini. Spre deosebire de KNN, care se bazeaza pe histograme, CNN-ul poate extrage informatii spatiale si de textura direct din imagine. Pentru antrenare s-au folosit tehnici de augmentare (flip, brightness, contrast, crop), iar modelul final a fost construit cu blocuri reziduale, stratificare convolutionala si regularizare L2. S-a folosit optimizatorul Adam cu scheduler CosineDecayRestarts si EarlyStopping pentru a evita overfitting-ul.

3.2 Proces de implementare

Ca si in cazul modelului KNN, prima etapa a constat in importarea librarilor necesare pentru implementarea si antrenarea modelului, precum si pentru citirea, prelucrarea imaginilor, manipularea datelor. In plus fata de bibliotecile standard folosite si in cazul modelului KNN (os, numpy, pandas, matplotlib etc.), pentru CNN a fost necesara utilizarea bibliotecii TensorFlow.

Modulele importate din TensorFlow includ:

- tensorflow.keras.layers si tensorflow.keras.models – pentru construirea arhitecturii retelei CNN, prin adaugarea de straturi convolutioanale, de pooling, dense etc.
- tensorflow.keras.utils.to_categorical– pentru conversia etichetelor claselor din format numeric intr-un format one-hot (adica vector binar) necesar pentru clasificarea multi-clasa . De exemplu eticheta 2 va fi convertita in $[0, 0, 1, 0, 0]$.

Pentru incarcarea si preprocesarea imaginilor am folosit o functie, similara celei utilizate anterior in abordarea cu KNN. Pentru claritate mentionez ca fiecare imagine este redimensionata la o dimensiune fixa (100x100 pixeli) si este normalizata prin impartirea valorilor pixelilor la 255 (exact ca in cazul modelului KNN).

Dupa ce datele au fost incarcate, acestea au fost pregatite pentru antrenarea modelului. Etichetele au fost convertite in formatul descris mai sus.

Apoi, seturile de date pentru antrenare, validare si testare au fost convertite in obiecte dataset. Datele au fost impartite in batch-uri de cate 64 de imagini (adica impartim setul mare de date in seturi mai mici a cate 64 de elemente fiecare - invata din 64 de imagini dintr-o data).

3.2.1 Primul Model - arhitectura simpla/experimentala

Prima versiune a modelului CNN a fost construita folosind o arhitectura simpla Sequential din Keras. Am folosit doar straturi de baza (convolutionale, pooling si dense), fara alte tehnici precum regularizare sau augmentare. Scopul acestei versiuni a fost sa inteleg mai bine cum functioneaza CNN-urile si sa pot compara direct performanta cu modelul KNN. Structura acestui prim model este urmatoare:

- Input: imagine de dimensiune 100x100x3 (conform cu structura RGB).
- Strat convolutional 1: aplica 16 filtre de dimensiune 3x3
- Pooling 1: reduce dimensiunea spatiala, pastrand informatia esentiala si reducand numarul de parametri (e implicit (2 2))
- Strat convolutional 2: aplica 32 filtre pentru a invata caracteristici mai complexe pe baza iesirilor anterioare. Dimensiunea este tot de 3x3.
- Pooling 2: reduce din nou dimensiunea spatiala
- Flatten: transforma rezultatul obtinut din straturile anterioare intr-un vector unidimensional necesar pentru straturile fully connected.
- Strat dens: reprezinta un strat fully connected cu 32 de neuroni
- Output: produce un vector de lungime 5, fiecare pozitie reprezentand probabilitatea ca imaginea apartine uneia dintre cele 5 clase.

Pentru straturile convolutionale s-a folosit functia de activare ReLU. Pentru stratul de iesire s-a utilizat softmax. Ea transforma vectorul de iesire intr-un set de probabilitati, astfel incat suma acestora sa fie 1. Clasa prezisa este cea asociata cu cea mai mare probabilitate. De exemplu daca avem vectorul [0.01, 0.02, 0.03, 0.04, 0.9] clasa prezisa este 4.

Compilare si antrenare

- Modelul a fost compilat cu:
  - SGD (Stochastic Gradient Descent) – un algoritm de optimizare simplu, care actualizeaza greutatile in directia care reduce eroarea. S-a folosit o rata de invatare fixa de 0.01.
  - categorical_crossentropy– functia de pierdere folosita pentru clasificare multi-clasa cu etichete one-hot.
  - accuracy – metrica folosita pentru a evalua performanta modelului.
- Antrenarea s-a facut pentru 20, 40 si 60 de epoci, cu afisarea progresului la fiecare epoca. La finalul fiecareia, modelul a fost evaluat si pe setul de validare.

CNN – 20 epoci - acuratetea pe validare a ajuns la 0.7192. Modelul nu e suprainvatat; acuratetea pe validare urmeaza indeaproape curba de antrenare. Matricea de confuzie arata o performanta decenta, dar clasa 4 este slab reprezentata (89 clasificari corecte, 113 confuzii in clasa 1). Totusi modelul pare ca nu a fost antrenat suficient.

CNN – 40 epoci - acuratetea pe validare ajunge la 0.7368.Curba de antrenare creste, iar cea de validare se mentine constanta, semn ca modelul invata si generalizeaza acceptabil. Matricea de confuzie se imbunatateste: clasa 4 are 106 clasificari corecte (fata de 89 anterior) si mai putine confuzii.

CNN – 60 epoci - acuratetea pe validare ajunge la 0.7376, graficul arata si aparitia overfitting-ului. Matricea de confuzie confirma: performanta pe clasa 4 nu se imbunatateste semnificativ (131 corecte, 63 confuzii in clasa 1).

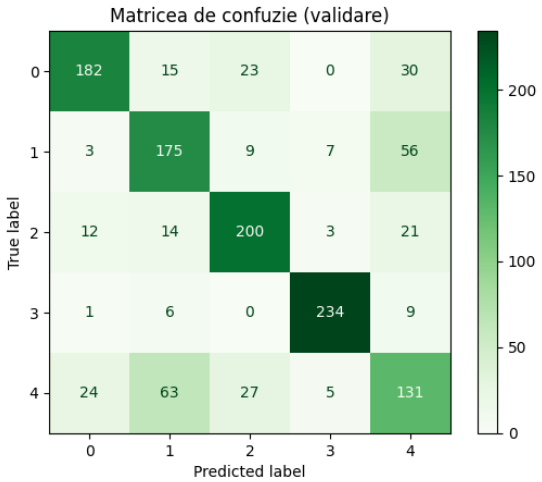

Figure 16: Model Simplu CNN - 60 epoci

Modelul CNN antrenat pentru 40 de epoci ofera un bun echilibru intre invatare si generalizare. Acuratetea este similara cu cea obtinuta dupa 60 de epoci. In continuarea experimentelor, am considerat rularea pentru 60 de epoci.

3.2.2 Imbunatatirea primului model - regularizare, augumentare, earlyStopping

Dupa prima varianta a retelei CNN, am observat ca, desi modelul invata bine pe setul de antrenare, acuratetea pe datele de validare nu era suficient de ridicata

si aparea un risc de overfitting. Astfel am aplicat treptat o serie de tehnici de imbunatatire a generalizarii/performantei. Mai intai am adaugat augmentarea datelor, pentru a creste diversitatea imaginilor de antrenare. Ulterior, am introdus EarlyStopping, pentru a opri antrenarea daca performanta pe validare nu mai crestea, evitand astfel suprainvatarea. Apoi am adaugat regularizare L2 in straturile dense si convolutionale, pentru a penaliza coeficientii mari si a controla complexitatea modelului. Am inlocuit optimizatorul SGD cu Adam, care se adapteaza automat la gradient si tinde sa ofere convergenta mai rapida si mai stabila. In final, am adaugat si un strat de Dropout, care dezactiveaza aleator neuroni in timpul antrenarii, fortand reteaua sa invete reprezentari mai robuste.

Parametrii pentru fiecare dintre aceste componente (ex: rata de invatare, coeficientul de regularizare, valoarea pentru dropout etc) au fost testati si ajustati treptat. Configuratia finala a oferit un echilibru intre performanta si stabilitate, cu o imbunatatire clara a acuratetii pe setul de validare fata de varianta initiala. In plus, am aplicat shuffle(1000) pentru a amesteca aleator datele si prefetch(tf.data.AUTOTUNE) pentru a incarca datele in avans, reducand astfel timpul de asteptare intre batch-uri. Mai jos am detaliat configuratia acestor metode.

- Augmentarea datelor – transformari random (flip orizontal, ajustari de contrast si luminozitate, crop) asupra imaginilor din setul de antrenare. Dupa acestea modelul a reusit sa generalizeze mai bine.
- Regularizarea L2 – pe straturile convolutionale si dense. Penalizeaza valorile mari ale ponderilor in timpul antrenarii. Cea mai optima valoare a fost de 0.0001 (am incercat si variante cu 0.001 / 0.00001)
- Dropout – am introdus un strat Dropout cu rata 0.3 inainte de ultimul strat dens. Se dezactiveaza aleator un procent din neuronii unui strat ca reteua sa nu depinda prea mult de anumiti neuroni
- Optimizator Adam – Am inlocuit SGD cu Adam, un algoritm de optimizare mai performant pentru retele neuronale. Cea mai buna pentru learning rate a fost 0.001.
- EarlyStopping – pentru a evita antrenarea inutila dupa ce modelul nu mai face progrese, am introdus un mecanism care opreste automat antrenarea daca pierderea pe setul de validare nu scade timp de 10 epoci consecutive. (am incercat si variante cu 6, 8, 12 pentru 60 de epoci, dar cele mai bune rezultate le-am obtinut pentru 10).

In urma acestor imbunatatiri, modelul a reusit sa obtina o acuratete mai mare pe datele de validare si sa mentina un comportament mai stabil pe parcursul antrenarii. Astfel am avut o acuratete finala de 0.8112. In plus, datorita mecanismului de EarlyStopping modelul s-a oprit la epoca 52/60.

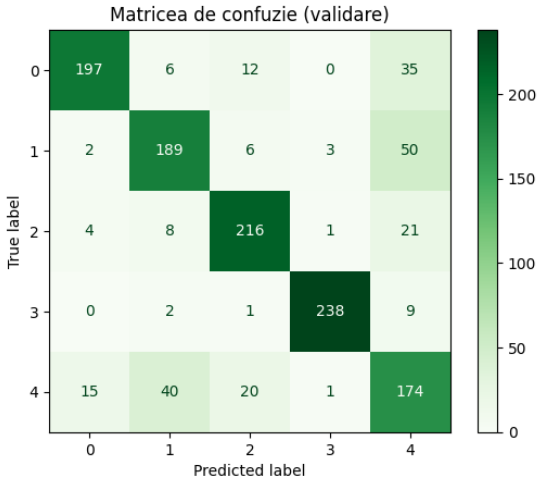

Figure 17: Model imbunatatit

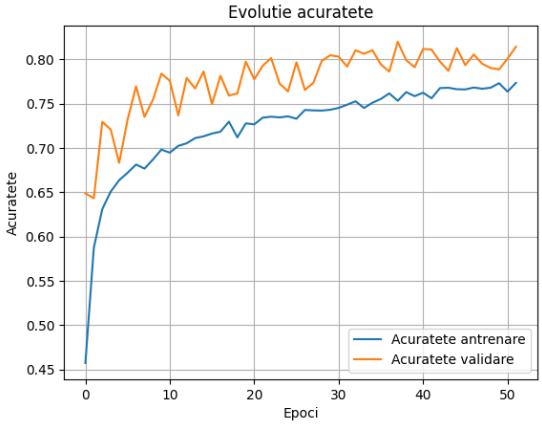

Figure 18: Model imbunatatit

Matricea ilustreaza urmatorul fapt: clasele 2 si 3 sunt clasificate destul de corect (cu peste 215 exemple etichetate bine), in timp ce cele mai multe confuzii apar, din nou, in cazul clasei 4. Totusi, chiar si in acest caz, modelul reuseste sa mentina o clasificare destul de precisa.

3.2.3 Al doilea model - cresterea complexitatii retelei

Mai departe, dupa ce am observat o imbunatatire a acuratetii dupa aplicarea tehnicilor mentionate anterior, am incercat sa optimizez si arhitectura retelei

propriu-zise. In acest pas, am folosit tehnici de normalizare precum BatchNormalization si am inlocuit stratul Flatten cu GlobalAveragePooling2D. Pe rand, am adaugat straturi suplimentare si am analizat impactul acestora asupra performantelor modelului.

Structura retelei este organizata in 3 blocuri principale:

- Primul bloc: doua straturi convolutinale cu 32 de filtre si BatchNormalization dupa fiecare, urmate de un strat MaxPooling2D.
- Al doilea bloc: doua straturi convolutinale cu 64 de filtre, fiecare urmat de BatchNormalization, urmate de MaxPooling2D.
- Al treilea bloc: un strat convolutinal cu 128 de filtre si BatchNormalization, urmat de MaxPooling2D.

La final, am inlocuit Flatten (varianta cu Flatten a obtinut o acuratete de 0.8136 pentru aceeasi retea) cu GlobalAveragePooling2D pentru a mai reduce din dimensiunea spatiala a datelor. Rezultatul este trimis apoi printr-un strat Dense cu 128 de neuroni si activare ReLU, urmat de un strat Dropout. In cele din urma, reteaua se incheie cu un strat Dense cu 5 neuroni si activare softmax.

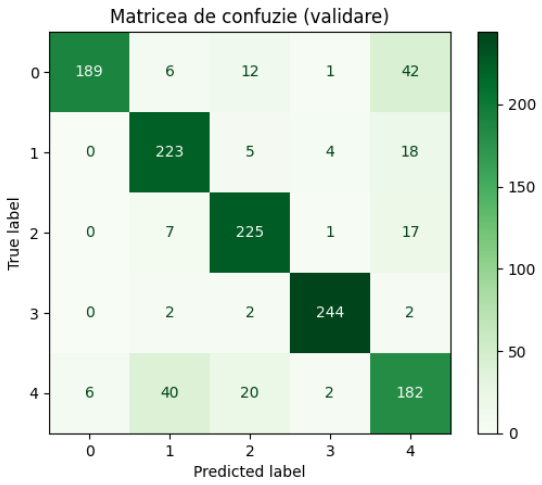

Figure 19: Model cu BatchNormalization

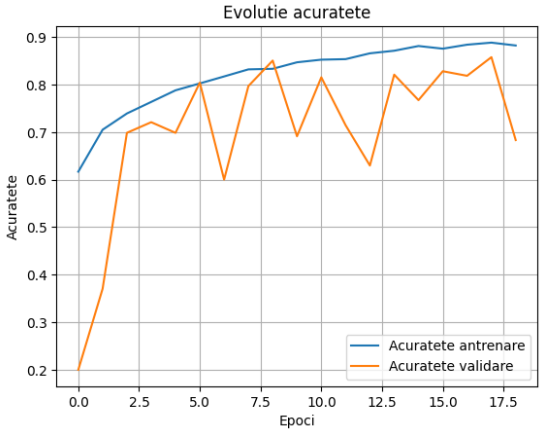

Figure 20: Model cu BatchNormalization

Modelul curent a atins o acuratete maxima de 0.8504 pe datele de validare, ceea ce reprezinta o imbunatatire fata de modelul anterior. Totusi, graficul de evolutie arata ca acuratetea pe validare oscileaza destul de mult. In schimb, acuratetea pe antrenare creste constant, semn ca modelul invata, dar nu este intotdeauna consistent pe date noi. Matricea de confuzie arata ca modelul face predictii bune pentru clasele 1, 2 si 3, insa clasa 4 este in continuare confundata mai ales cu clasa 0.

Comparativ, modelul anterior era mai simplu si avea o evolutie mai stabila, insa performanta era usor mai scazuta. Astfel, am continuat perfectionarea modelului curent.

3.2.4 Imbunatatirea celui de al doilea model - scheduler

Pentru a imbunatati modelul, am incercat sa adaug un scheduler (un mecanism care modifica automat rata de invatare in timpul antrenarii). In locul unei valori fixe (0.001), am utilizat o alta strategie, care ajusteaza progresiv rata de invatare in functie de epoca. Tipul de scheduler ales a fost CosineDecayRestarts.

Acest scheduler reduce treptat rata de invatare pe baza unei functii cosinus, cu resetari periodice, permitand retelei sa continue invatarea eficienta. Dupa mai multe teste, urmatorii parametri s-au dovedit cei mai potriviti pentru acest model:

- initial_learning_rate = 1e-3- rata de invatare initiala este 0.001.
- first_decay_steps = 10×196 - numarul de pasi (batch-uri) pentru primul ciclu este echivalentul a 10 epoci (196 fiind numarul de batch-uri pe epoca - 12500/64 aproximativ 196).

- t_mul = 2.0 - fiecare ciclu urmator este de doua ori mai lung decat precedentul.
- m_mul = 0.9 - dupa fiecare ciclu, rata maxima de invatare scade cu 10%. (100%-90%)
- alpha = 1e-4 - valoarea minima a ratei de invatare in interiorul unui ciclu este 0.0001.

Astfel, am obtinut o acuratete de 0.8720 – o imbunatatire semnificativa fata de varianta fara scheduler, care a obtinut o acuratete de 0.8136 pentru aceeasi constructie a retelei.

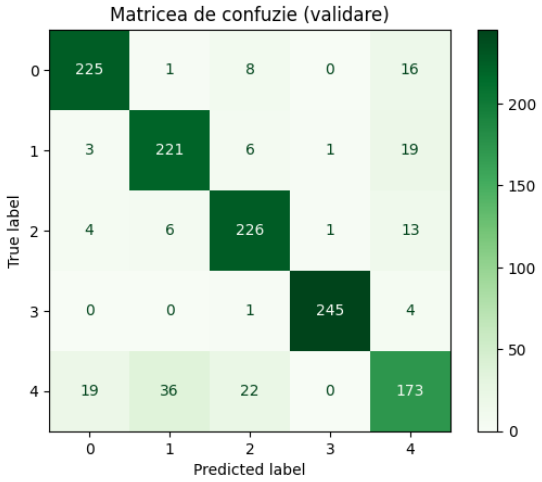

Figure 21: Model cu CosineDecayRestarts

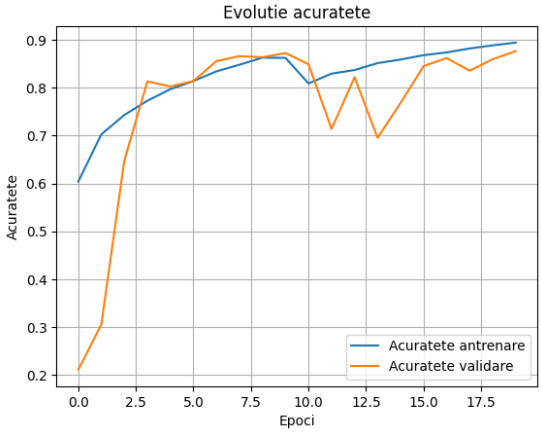

Figure 22: Model cu CosineDecayRestarts

Modelul antrenat cu scheduler CosineDecayRestarts a obtinut cele mai bune rezultate dintre toate versiunile testate. Graficul arata o crestere rapida a acuratetii in primele epoci, urmata de o evolutie stabila, semn ca scheduler-ul a ajutat la reglarea ritmului de invatare. Matricea de confuzie indica faptul ca majoritatea claselor sunt recunoscute corect (peste 220 de exemple in clasele 0–3), iar erorile sunt mai bine distribuite. Clasa 4, care anterior avea cele mai mari probleme, este acum tratata mai eficient, dar prezinta in continuare confuzii.

3.2.5 Al treilea model

Pentru modelul final, am optat pentru o arhitectura bazata pe blocuri reziduale. Acest tip de structura s-a dovedit eficient in antrenarea retelelor neuronale mai adanci, deoarece ajuta la transmiterea informatiei si a gradientului intre straturi.

Principalul avantaj al blocurilor reziduale este ca introduc o conexiune directa (scurtatura) care permite retelei sa ”sara” peste unele transformari, daca acestea nu contribuie semnificativ la performanta. In acest fel, reteaua poate invata mai usor doar transformarile esentiale, iar in caz contrar, informatia initiala este transmisa mai departe fara modificari.

Descrierea retelei:

- Blocul 1 – 32 de filtre - se aplica o convolutie initiala cu 32 de filtre si activare ReLU. Rezultatul este salvat in variabila scrt, care va fi scurtatura. Apoi se aplica doua straturi Conv2D cu 32 de filtre si BatchNormalization (asemanator cu arhitectura precedenta). Rezultatul este combinat in

cele din urma cu scurtatura. Se aplica MaxPooling2D pentru reducerea dimensiunii spatiale.

- Blocul 2 – 64 de filtre - scurtatura este trecuta printr-o convolutie 1x1 . Se aplica doua straturi Conv2D cu 64 de filtre si BatchNormalization. Se face adunarea cu scurtatura modificata si se aplica MaxPooling2D.
- Blocul 3 – 128 de filtre - scurtatura este din nou ajustata cu o convolutie de 1x1. Se aplica doua straturi Conv2D cu 128 de filtre, fiecare urmat de BatchNormalization. Rezultatul este adunat cu scurtatura si apoi se aplica GlobalAveragePooling2D.
- Final - se aplica un strat Dense(128) cu activare ReLU si regularizare L2, dupa care se adauga un Dropout de 0.5. Urmeaza un strat de iesire Dense(5) cu activare softmax, care genereaza predictii pentru cele 5 clase.

Acuratetea obtinuta pentru acest model a fost si cea mai buna de pana acum, anume 0.9224 pe datele de validare.

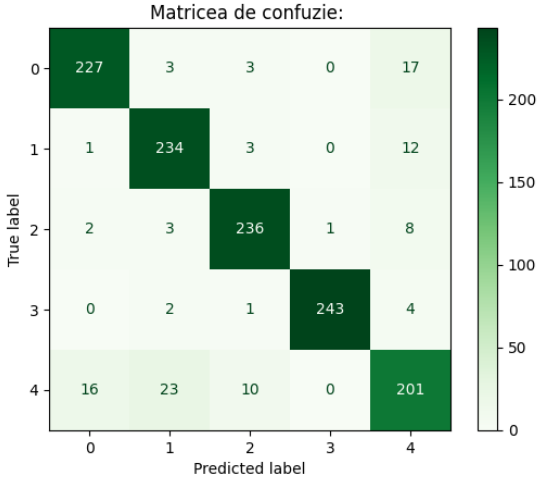

Figure 23: Model Blocuri Reziduale

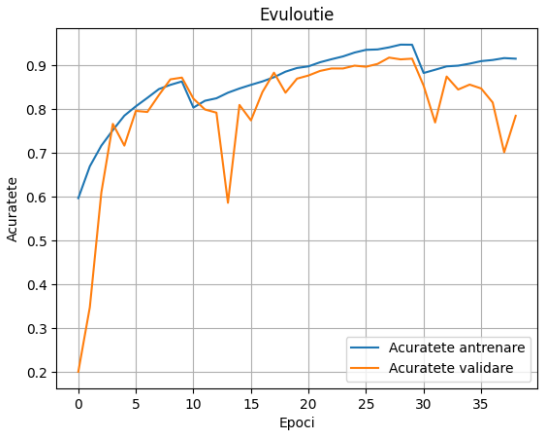

Figure 24: Model Blocuri Reziduale

Graficul arata ca acuratetea pe setul de antrenare a crescut constant si a atins valori peste 92%, ceea ce sugereaza ca modelul a invatat bine datele. In ceea ce priveste matrice de confuzie, clasele 0, 1, 2 si 3 sunt recunoscute foarte bine, cu peste 230 exemple corecte fiecare. Clasa 4 este in continuare mai slab recunoscuta comparativ cu celelalte, dar s-a imbunatatit fata de modelele anterioare.

Modelul cu blocuri reziduale, regularizare, augmentare si scheduler a obtinut cele mai bune performante de pana acum. Acuratetea pe validare a urcat la aproximativ 0.92, iar modelul nu prezinta semne evidente de overfitting. Matricea de confuzie arata o distributie buna a predictiilor.

3.3 Concluzie generala asupra modelelor CNN

- Modelul 1 (de baza) a fost o retea CNN simpla, cu doua straturi convolutionale urmate de MaxPooling si Dense. Acesta a atins o acuratete modesta de 0.7376, fiind un punct de pornire pentru experimentele ulterioare.
- Modelul 2 a introdus augmentarea datelor, regularizarea L2, Dropout si BatchNormalization, ducand la o crestere vizibila a acuratetii de 0.8504. A fost folosita o arhitectura mai adanca, cu mai multe straturi si GlobalAveragePooling. Modelul a devenit mai stabil si a generalizat mai bine.
- Modelul 2 imbunatatit (cu scheduler) a folosit aceeasi arhitectura ca modelul anterior, dar a inlocuit rata de invatare fixa cu un scheduler de tip CosineDecayRestarts. Acest mecanism a imbunatatit convergenta si a dus la o acuratete mai mare de 0.8720, cu rezultate mai stabile pe validare.

- Modelul final (cu blocuri reziduale) a utilizat o arhitectura avansata, care aduce in plus blocuri reziduale. Acest model a atins cea mai buna performanta: acuratete 0.9224 pe validare.

4 Concluzii finale

In cadrul proiectului am testat modele diferite pentru clasificarea imaginilor. Prima directie a fost data de un model clasic KNN bazat pe histograme, care a oferit rezultate modeste, fiind limitat de simplitatea caracteristicilor extrase. A urmat un model CNN imbunatatit, cu straturi convolutinale, normalizare si regularizare, care a dus acuratetea cu mult mai buna . In final, am construit un model CNN cu blocuri reziduale si scheduler pentru learning rate, care a obtinut cea mai buna performanta: aproximativ 0.9224 pe validare. Fiind singurul model care a depasit pragul de 0.9 acesta a fost ales pentru submiterea finala in competitie. In cele din urma, modelul final a obtinut o acuratete de 0.9208 pentru datele de test in competitia Kaggle.

5 Note privind codul

Au fost incarcate cele mai importante si relevante puncte din procesul de dezvoltare, precum si cele care au oferit cele mai bune rezultate la acel moment. Codul pentru fiecare incercare descrisa in document se afla in urmatoarele fisiere:

- Modelul final KNN (sectiunea 2.4): KNN.py
- Primul model CNN – varianta de baza a retelei + optimizari (sectiunea 3.2.2): CNN_1.py
- Al doilea model CNN – varianta imbunatatita a retelei si adaugarea schedulerului (sectiunea 3.2.4): CNN_2.py
- Al treilea model CNN – modelul final cu blocuri reziduale (sectiunea 3.2.5): CNN_final.py

Fisierele .py pentru modelul CNN includ toate variantele relevante ale retelei.