UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare

DEEPFAKE CLASSIFICATION

Student

Roscan Rares-Alexandru

github

Coordonator științific

Prof.dr. Radu Ionescu

București, iunie 2025

Cuprins

Capitolul 1

Setul de date

Tema: Clasificarea imaginilor deepfake generate de retele neuronale adanci. Obiectivul principal: Construirea unui model capabil să distingă automat între imaginile autentice (reale) și imaginile deepfake. Este vorba despre un set de date generos si etichetat, care contine 12500 de imagini pentru antrenare, 1250 de imagini pentru validare si 6500 de imagini pentru testare. Setul de date este disponibil public pe Kaggle, iar imaginile sunt etichetate ca apartinand unei categorii, de la 0 la 4.

Astfel dupa ce am analizat cateva exemplare pentru fiecare eticheta in parte am realizat ca este vorba despre niste imagini ”animate”.

Urmatorul pas a fost verificarea dimensiunii setului de date. Dupa rularea unui script care verifica dimensiunea tuturor imaginilor, am ajuns la concluzia ca toate acestea respecta formatul de 100:100 pixeli (ceea ce inseamna ca nu este necesara redimensionarea lor).

A doua verificare a fost daca existe poze alb-negru, deoarece acest lucru ar fi putut afecta performanta modelului. Dupa rularea unui script care verifica daca exista poze alb-negru, am ajuns la concluzia ca nu exista astfel de poze in setul de date.

Astfel am tras concluzia ca setul de date este unul de calitate si curat, ceea ce este un lucru pozitiv pentru antrenarea modelului.

Capitolul 2

Pregatirea setului de date si primele experimente

Pentru inceput am decis ca as dori sa verific 2 algoritmi de clasificare, si anume: **SVM** si **Random Forest**.

Avand in vedere ca setul de date este unul curat si foarte accesibil, am decis sa verific care ar fi rezultatele obtinute de cei 3 algoritmi de clasificare mentionati. Stiam ca algoritmul **Random Forest** este robust si ca este foarte posibil ca acesta sa aiba rezultate foarte bune direct pe pozele in formatul lor original (adica cel .png), dar am decis sa realizez mai inati normalizarea setului de date, deoarece ar fi fost necesar pentru algoritmul **SVM**.

Am calculat media si deviatia standard a setului de date, si am realizat normalizarea acestuia. Dupa ce am realizat normalizarea, am decis sa incerc algoritmul **Random Forest** pe imaginile originale. Rezultatele obtinute au fost promitatoare:

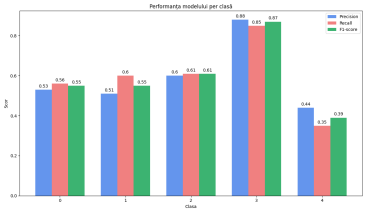


Figura 2.1: Rezultatele obtinute in urma procesului de antrenare si validare

Al doilea pas al experimentului a fost antrenarea modelului pe datele normalizate salvate in format .npy. Am realizat acest lucru deoarece am vrut sa vad daca normalizarea setului de date are un impact pozitiv asupra rezultatelor obtinute, dar dezamagitor rezultatele au fost identice cu cele obtinute in urma antrenarii modelului pe datele originale:

Rezultatele fiind aproape identice am ajuns la concluzia ca normalizarea setului de date nu are un impact asupra rezultatelor obtinute de algoritmul **Random Forest**, si ca acesta este un algoritm robust care poate fi folosit direct pe datele originale. Cu toate

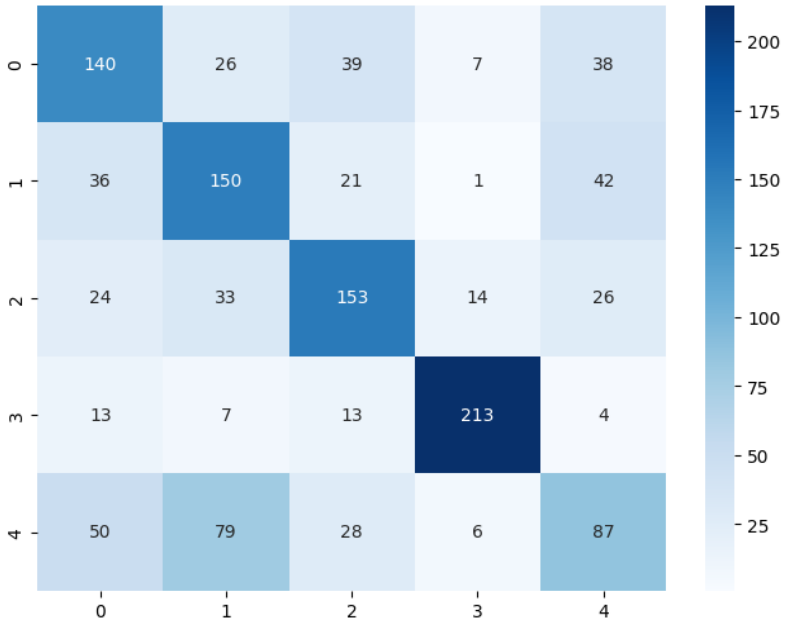


Figura 2.2: Matricea de confuzie obtinuta in urma procesului de antrenare si validare

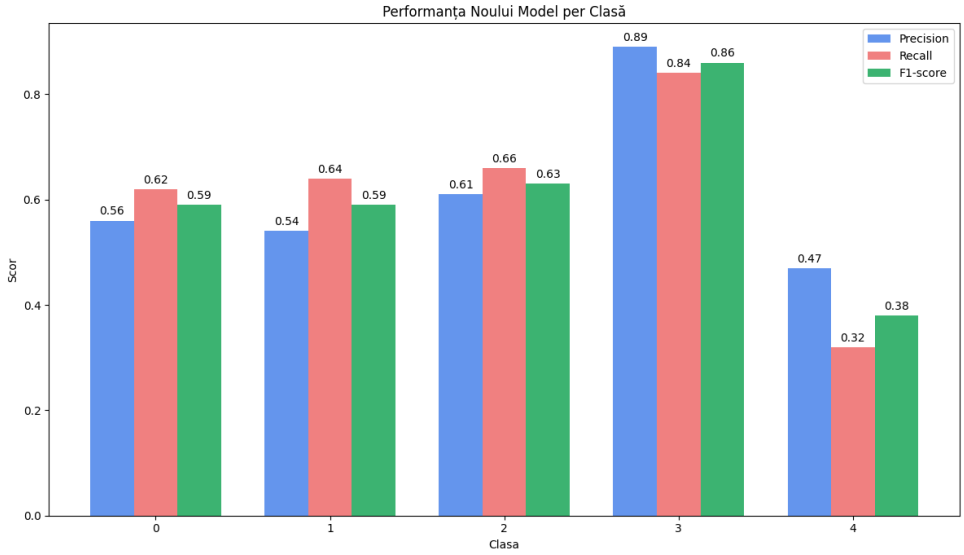


Figura 2.3: Rezultatele obtinute in urma procesului de antrenare si validare pe datele normalizate

acestea am decis ca rezultatele lasa de dorit si nu am insitat prin fine-tuning (ajustarea hiperparametrilor), scopul fiind de a ajunge la un scor de minim 90% acuratete.

Dupa experimentul cu algoritmul **SVM**, am observat ca trendul continua:

Cu toate acestea rezultatele obtinute au fost foarte slabe in continuare, si am considerat ca deocamdata nu are sens modificarea hiperparametrilor fara sa inteleg mai bine setul de date.

Ce am inteles totusi este ca ambii algoritmi nu au avut capacitatea de a intelege

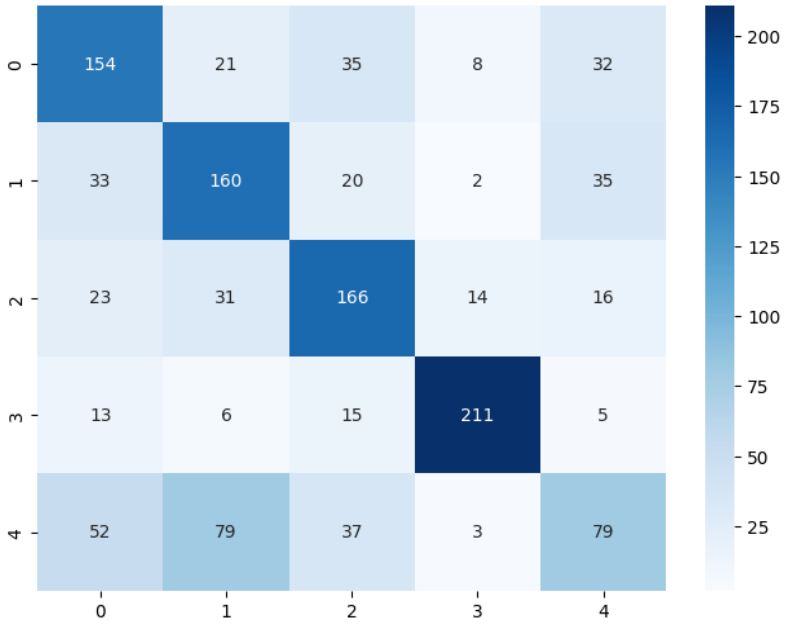


Figura 2.4: Matricea de confuzie obtinuta in urma procesului de antrenare si validare pe datele normalizate

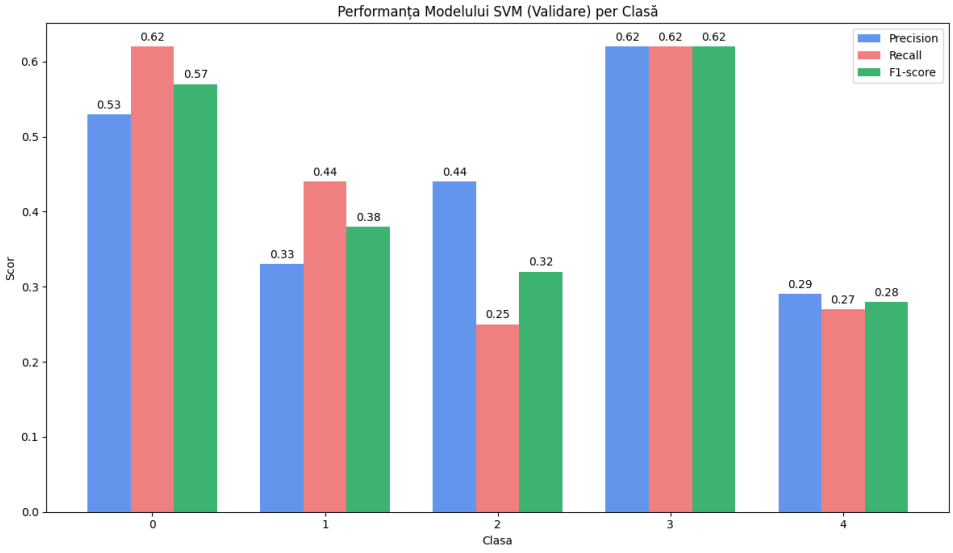


Figura 2.5: Rezultatele obtinute in urma procesului de antrenare si validare cu algoritmul SVM

imaginile doar pe baza pixelilor; acestia nu au capacitatea de a capta texturi, posibile artefacte si alte nuante care ar putea fii definitorii clasificarii noastre.

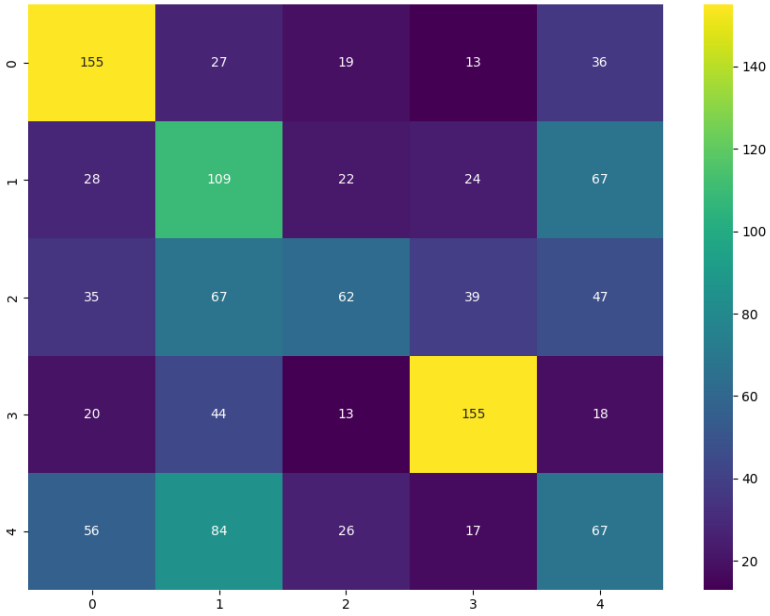


Figura 2.6: Matricea de confuzie obtinuta in urma procesului de antrenare si validare cu algoritmul SVM

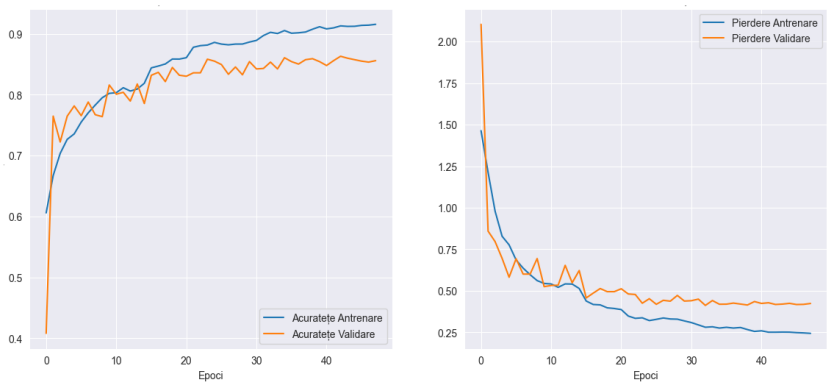
Capitolul 3

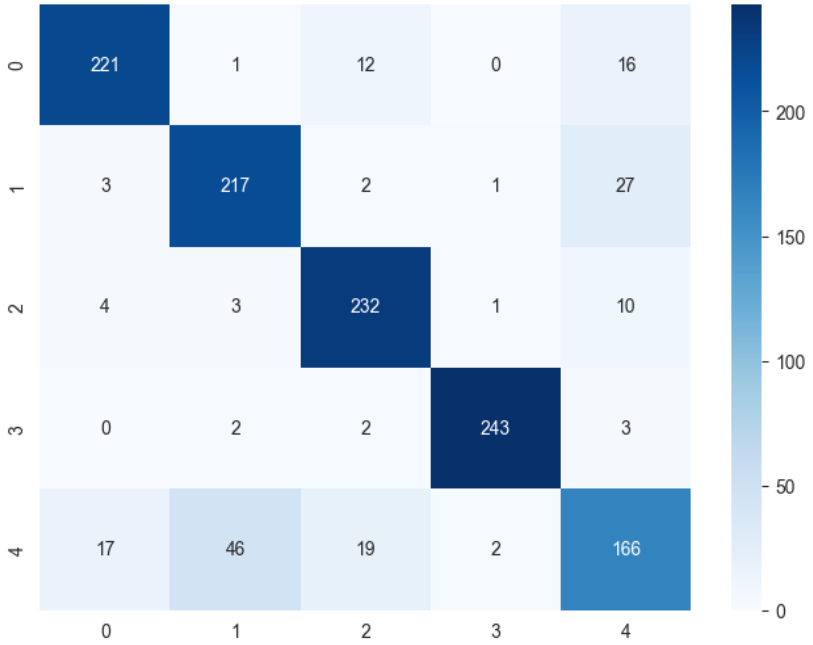
Retelele convolutionale

Dupa rezultatele obtinute am decis sa ma documentez mai bine in legatura cu image vision. Am descoperit un concept numit **Retea Convolutionala**. Am urmarit tutorialul de pe TensorFlow si am inceput sa ma joc cu acest concept. Motivul principal pentru care am considerat ca ar fi superioara aceasta metoda, a fost faptul ca retelele convolutionale sunt special concepute pentru a procesa datele de tip imagine, avand in vedere ca acestea pot extrage caracteristici importante din imagini precum texturi sau alte artefacte (ceea ce ma asteptam sa fie absolut necesar pentru poze deepfake).

Dupa ce am rulat N experimente cu diferite structuri de retele am observat diferite aspecte:

1. La prima vedere sunt greu de antrenat, deoarece au foarte multi hyperparametri (de la numarul de straturi, la numarul de neuroni, viteza de invatare, etc.). Pentru ca era vorba de foarte multe epoci (peste 20-30 de epoci inca de la primele experimente), am concluzionat ca este aproape imposibila salvarea manuala si rularea de 5-10 ori a aceleiasi retele pentru a gasi cel mai bun model (adica puteam sa rulez pana la 25 de epoci, dar cel mai bun model era intre epocile 20-25). M-am documentat si am descoperit modulul callbacks intr-o postare pe reddit. Astfel am implementat early stopping, reducerea vitezei de invatare si salvarea modelului cel mai bun in timpul antrenatii.





2. Al doilea pas a fost incercarea diferitelor structuri de retele. Prin utilizarea de callbacks, am reusit sa salvez modele mai puternice (asta datorita reducerii vitezei de invatare).

Am incercat retele cu 3 si 4 straturi pe diferite latimi, si am observat ca in general se ajungea la overfit.

De fiecare data am avut probleme cu clase 0, 1, 2 catre clasa 4. Clasa 3 a fost identificata corect. In locul clasei 4 erau atribuite celelalte clase mentionate.

3. Am incercat data augmentation prin rotire, zoom si flip, dar rezultatele au fost mai slabe. A incercat zoom de la 10% pana la 1%, dar modelul pierdea din acuratete, iar loss-ul crestea foarte mult. Astfel am decis sa nu mai folosesc data augmentation, deoarece am considerat ca modelul are anumite caracteristici care erau ”sterse” de data augmentation.

4. Am inceput sa rulez mai multe teste pentru a intelege mai bine imaginile. Deoarece modelul de **Random Forest** a avut capacitatea de a prezice cu 70% acuratete doar pe baza pixelilor, am decis sa verific canalele de culoare.

Dupa cele 3 verificari, am decis ca distributia HSV pare a fi inutila, pe cand cea LAB avea potential de a fi mai capabila. Am rulat modelul pe distributia LAB, si am avut rezultate similare cu distributia pe RGB. Dupa am incercat doar primul canal LAB, deoarece vizual mi s-a parut ce fiind cel cu diferentele cele mai mare, dar rezultatele au scazut mult.

Am mai realizat un test, si anume plotarea culorilor folosind algoritmul **t-SNE**, dar

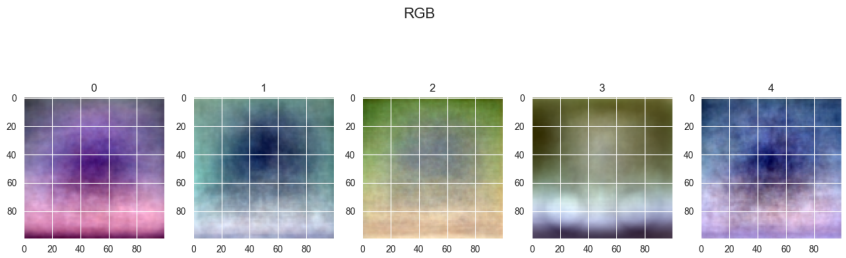


Figura 3.1: Distributia RGB

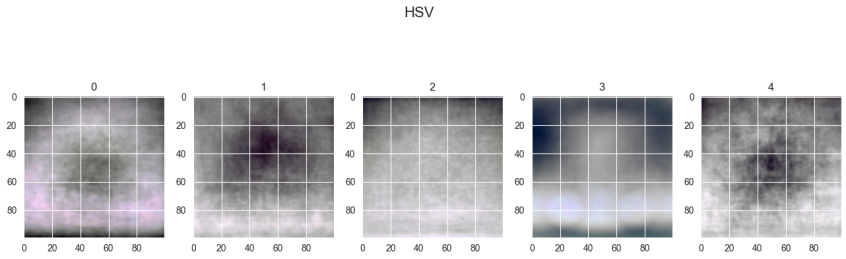


Figura 3.2: Distributia HSV

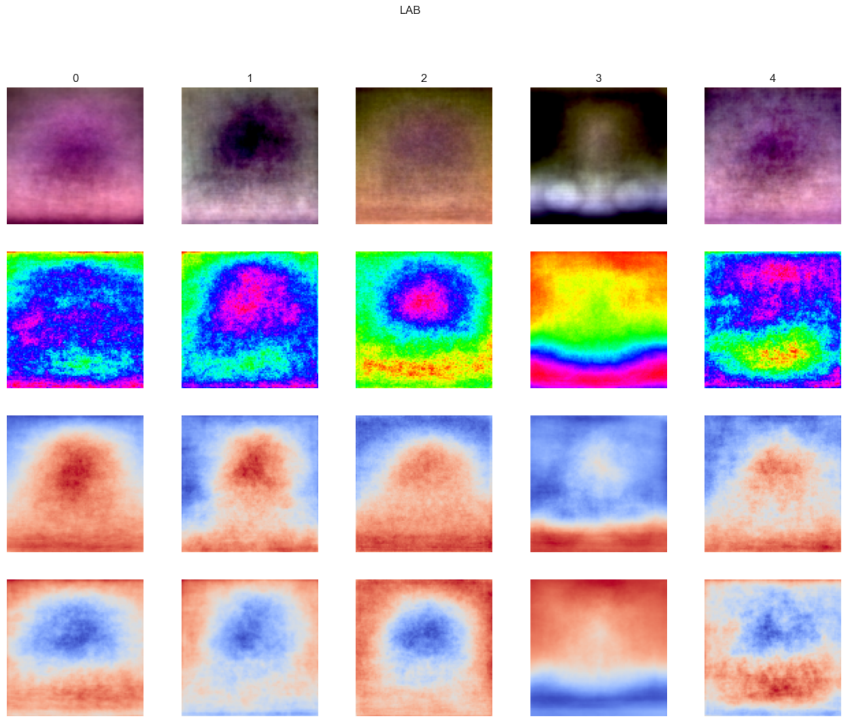
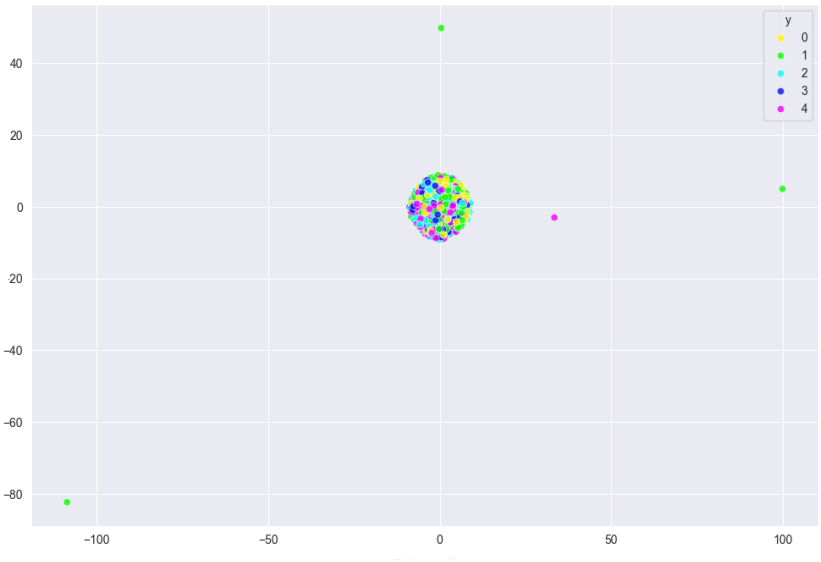
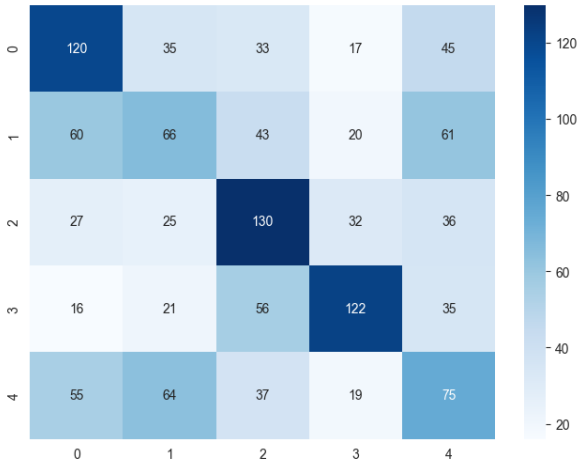


Figura 3.3: Distributia LAB





totul era ingramadit si nu se observa potentialul de separare al imaginilor pe etichete.

5. Am incercat sa antrenez un model **XGBoost** pe baza histogramelor de culori, deoarece canalul RGB cel putin parea a avea potential. Rezultatele au fost foarte slabe, si am ajuns sa renunt complet la ideea de a folosi orice altceva in afara unei retele convolutionale.

6. Am revenit la retelele convolutionale fara data augmentation. Am incercat toate varaintele posibile, atat in adancime cat si in latime. Am incercat sa modific viteza de invatare, sau parametrii din callbacks, dar nu am reusit sa obtin mai mult de 91.4% acuratete pe validare. Mereu aveam overfit mare, acuratetea pe antrenare ajungea la 97%, si la validare cu greu trecea de 90%. Astfel am inceput sa ma documentez in legatura cu data augmentation.

Am descoperit o tehnica noua, si aceea fiind **Cutout**. Nu am avut mari sperante, dar ar fi fost prima tehnica incercata, care nu ar fi modificat ”ordinea pixelilor din imagine”.

Am intuit ca rezultatele slabe ale zoom-ului, rotirilor si flip-urilor ar fi fost cauzate din cauza modului in care s-a produs generarea, iar cutout-ul ar fi fost o abordare "oarecum" diferita.

Astfel, am reusit sa ajung la **Underfit**, dupa zeci, daca nu sute de teste rulate, acuratetea pe antrenare era 70% si pe validare 80%. Ultimul impediment a fost rularea antrenamentului. Zic asta deoarece orice alt antrenament realizat anterior a durat pana la maxim 20 de minute (pentru maxim de 45 de epoci).

De data asta vorbim despre 7 epoci, care au durat 1 ora si 15 minute.

Dupa am incercat si **CutMix** care a avut rezultate si mai bune. Problema mea a fost underfit-ul in continuare. Am incercat treptat mai multe arhitecturi de retea, am pornit de la 3 straturi, iar in ultima iteratie (unde am obtinut locul 11 in clasament) am folosit 5 straturi si late. **mentionez ca inca aveam underfit**. Ultima mea varianta a fost antrenata pe un maxim de 100 de epoci deoarece nu mai aveam timp (adica, as fi avut o acuratete mai mare); mai aveam 2 minute in momentul in care s-a terminat antrenarea pana la deadline-ul competitiei.

Deoarece am rulat atat de multe experimente, am pierdut pasul cu plotarea rezultatelor, iar rularea testelor din nou pentru a obtine graficele ar dura o vesnicie pe calculatorul meu.