**Dinu Matei-Alexandru, 352**

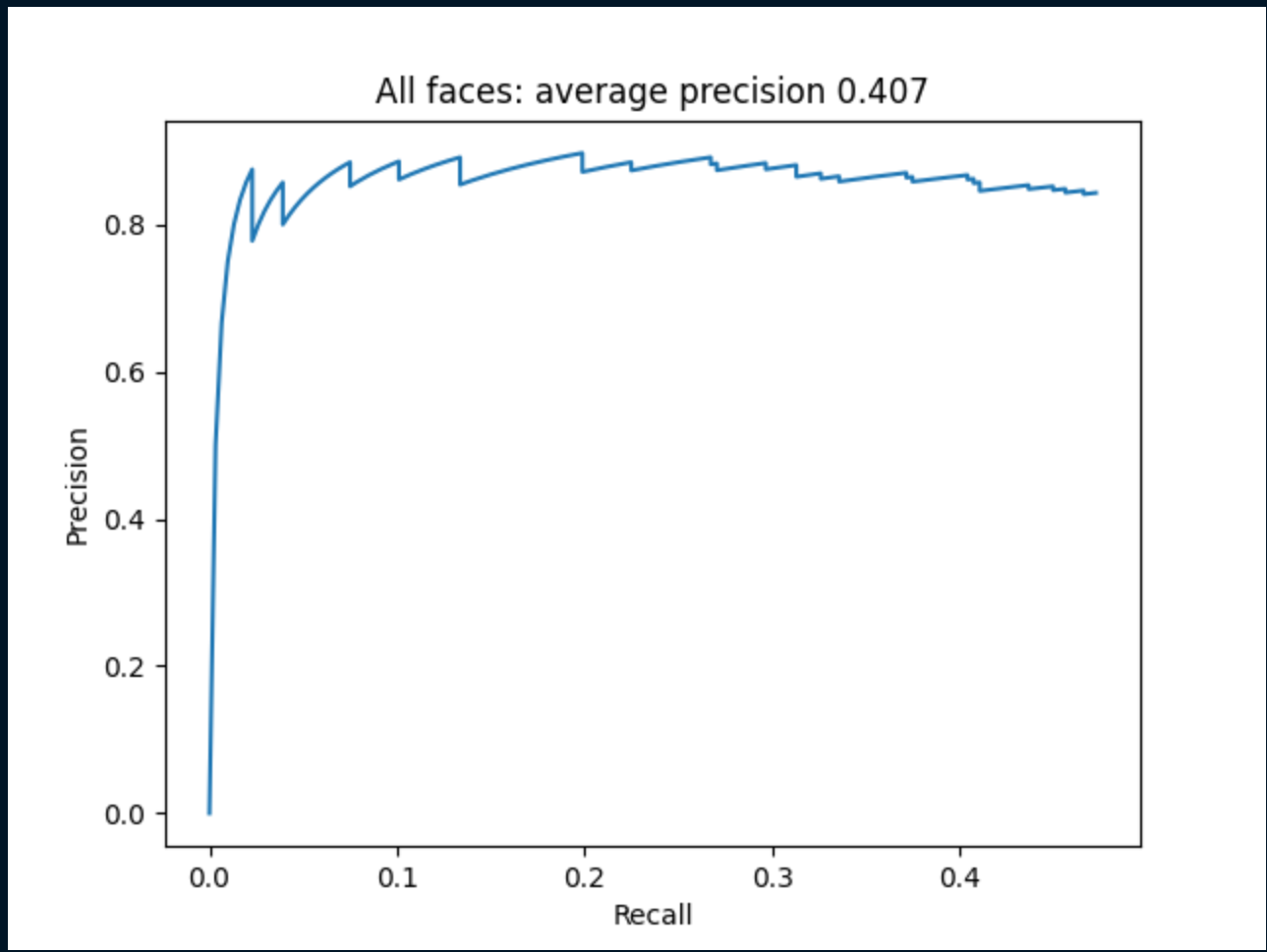
**Dexter Face Detector**

Proiectul presupune detectia fetelor prezente in fiecare imagine. Initial, am abordat task-ul 1, deoarece scorul pentru task-ul 2 depinde foarte mult de task-ul 1. Initial, am format datele. M-am asigurat ca toate fisierele sunt intr-un singur folder si am cropat astfel incat in fiecare imagine sa fie exclusiv o fata. Label-urile pentru fiecare fata le-am luat din denumirea fiecarei imagini. Practic, am gasit o oarecare ambiguitate in denumirea initiala a imaginilor deoarece, de exemplu, intr-o imagine label-uita ca “dad” puteau sa apara si alte personaje, asada­r in functie de fisierul cu ground truths pentru datele de train, am creat imagini care sa reflecte exclusiv fetele fiecarui personaj. Pentru asta, am creat fisierul DatasetCreationPlusBounding.py, care face structura folderelor facila pentru antrenat modelul CNN de recunoastere (prezentat ulterior), dar si fisiere suplimentare pentru o mai buna organizare a datelor. Aceasta impartire este mai usoara deoarece face mai facila codul care nu trebuie sa contina path-uri sau preprocesari suplimentare.

Apoi, urmeaza procesarea descriptorilor HOG. Am ales dimensiune 9 pentru features si am mentinut dimensiunea standard din laborator de 3 cells per block, cu 10 orientari. Am incercat cu dimensiuni variate ale hog features, in mare parte trial and error, unii descriptori ocupau extrem de mult spatiu RAM, iar daca nu foloseam partea de stocare swap antrenarea modelului nu mai functiona deoarece sistemul pe care faceam train-ul ramanea fara RAM. Am descoperit aceste valori ca fiind “echilibrate”. Normalizarea a fost lasata la fel. Ca dimensiuni de procesare, am incercat sa mentin un ratio de 1:5 intre exemple pozitive si cele negative. Am incercat si cu ratio de 1:1, dar diferenta intre detectiile pozitive si cele negative era prea mica. Pentru fiecare imagine pozitiva, am adaugat si imaginea rotita pentru a mari numarul total de exemple pozitive la ~11000. Numarul de exemple negative este de ~446 in total, si am folosit un set de patch-uri random din fiecare imagine pentru a scoate hog features din ele si a le adauga la descriptorii negativi. Am extras exemplele negative din secvente random din desenul Dexter. Pentru descriptori, am considerat feature-uri pentru dimensiuni variate ale ferestrei (random in negative) si pozitive, astfel ca am inmultit numarul de exemple pozitive la aproximativ 20000 (in total, extrase din toate marimile ferestrelor) si cam 100000 de exemple negative. Pentru fiecare din aceste imagini, procesam descriptorii, ii incarcam in RAM, apoi facem train pe modelul de clasificare fata/non fata pe fiecare fereastra din sliding window.

Ca model de clasificare, incercarea initiala a fost cu un LinearSVM, cu valori variate pentru C, max\_iterations variat, dar nu am obtinut scoruri prea bune, apoi cu un Linear SVM cu kernel rbf, unde nu am obtinut rezultate bune la detectii, erau destul de multe portiuni din imagine detectate gresit, care nu puteau fi “scoase” dupa rularea non maximal suppression. Cel mai bun rezultat l-am obtinut cu un model SGD cu hinge loss. Am aplicat normalizare, pentru a avea medie 0 si varianta 1, apoi am folosit batch-uri de cate 2000 de elemente, cu 20000 de iteratii, am folosit regularizare foarte mica prin parametrul alpha. Aici am riscat sa creez overfitting, dar, folosind partial fit pentru invatare incrementala pe date random (15 epoci, in care luam feature-uri random cu label-uri random, apoi facem partial fitting pe ele, pastrand la fiecare iteratie cea mai buna performanta a modelului). Am descoperit ca partial fitting si invatarea incrementala se potriveste mai bine in contextul nostru. In general, am oscilat parametri modelului. Am folosit class weight dict pentru a atribui greutati mai mai potentialelor clase imbalansate, am folosit parametrul power\_t pentru a mentine learning rate-ul ridicat pentru perioade mai lungi de timp, permitand modelului sa faca ajustari mari si in iteratii mai tarzii, practic oferind mai mult timp sa invete, iar warm\_start permite modelului sa isi salveze starea precedenta pentru partial fitting.

La fiecare epoca si iteratie, salvez modelul cu scorul cel mai bun din batch-ul curent. Dupa ce modelul si feature-urile normalizate au fost salvate pentru modelul cel mai bun, iar datele de validare au fost incarcate, putem rula mecanismul de sliding window. Pentru asta, am generat dimensiuni variate pentru a verifica modelul. Am inceput cu un array generat logaritmic, pentru a prioritiza dimensiunile mici (step-ul este mai mic la ferestre de dimensiuni mici), iar apoi, pentru fiecare dimensiune am setat cate un aspect ratio, pentru a genera aproximativ 150 de ferestre cu latimi si inaltimi diferite, iar daca numarul lor depaseste 150 (pentru a mentine un timp de rulare rezonabil), le scoatem pe cele mult prea late sau mult prea inalte sau mari si prioritizam ferestrele mai mici. Trecem imaginea in spectrul grey, apoi facem sliding window cu fiecare din ferestrele rezultate, pentru fiecare fereastra calculam si varianta, iar daca este prea mare sarim peste ea, apoi redimensionam fiecare fereastra la 64x64 pentru a putea calcula hog features, care sunt normalizate si trecute prin model. Daca modelul atribuie un scor mai mare decat un threshold, atunci avem o detectie valida, iar pentru fiecare din aceste detectii aplicam non maximal suppression pentru a mentine doar ferestrele locale cele mai favorabile, punem coordonatele ferestrelor ramase intr-un fisier text, dar generam si o imagine cu dreptunghiuri in jurul detectiilor, plus scorul atribuit de model. Pentru partea de clasificator, pentru a imbunatati performanta am folosit 12 procese paralele, iar pentru a monitoriza progresul general am folosit biblioteca tqdm, pentru loading bar. La acest task, cu ultimul model, am obtinut o precizie de 0.352 pe datele de validare, aceasta valoare osciland in functie de valorile random, de exemplu ce patch-uri se iau pentru hog features, ce batch-uri random se considera pentru model, etc.



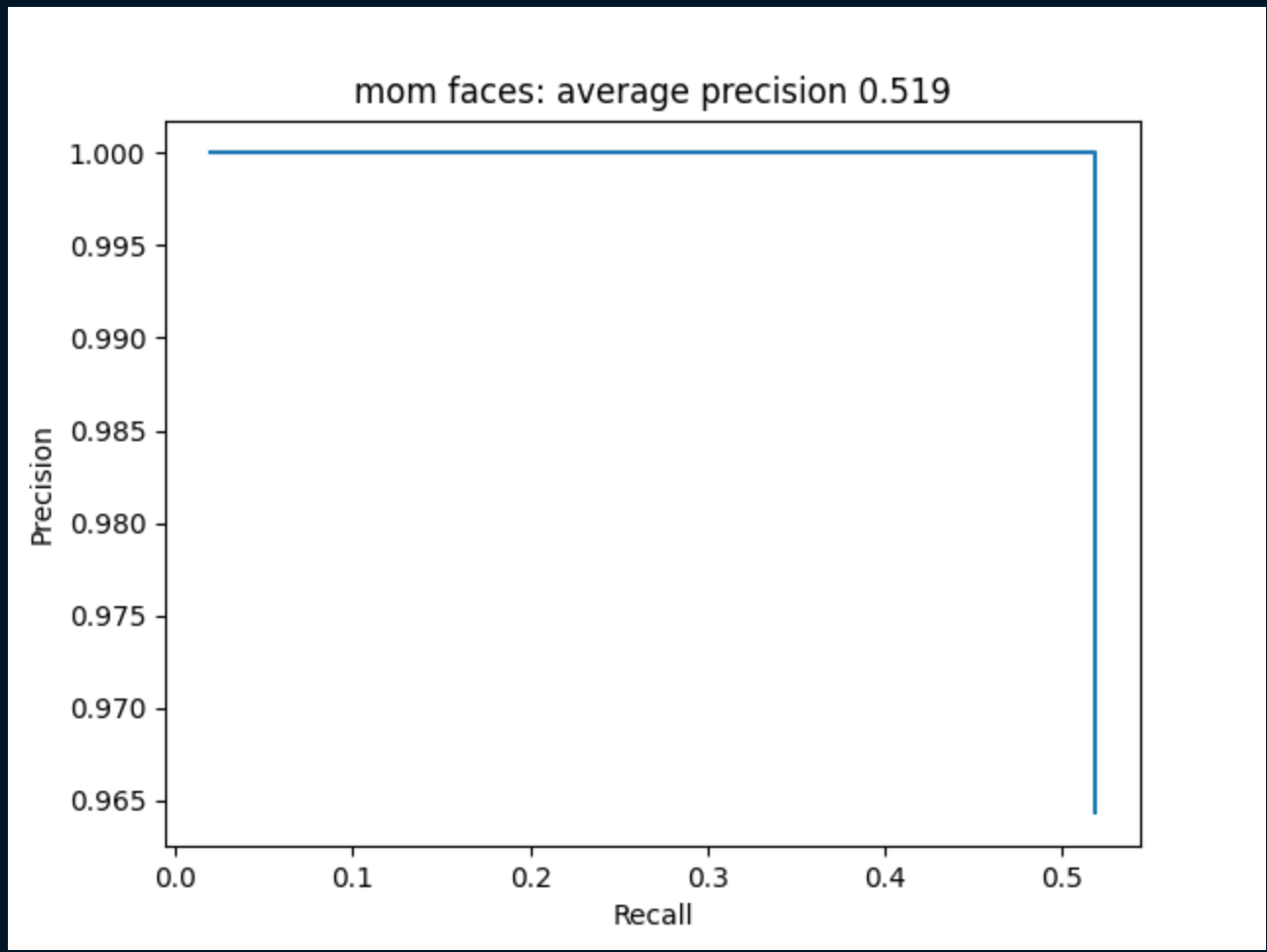
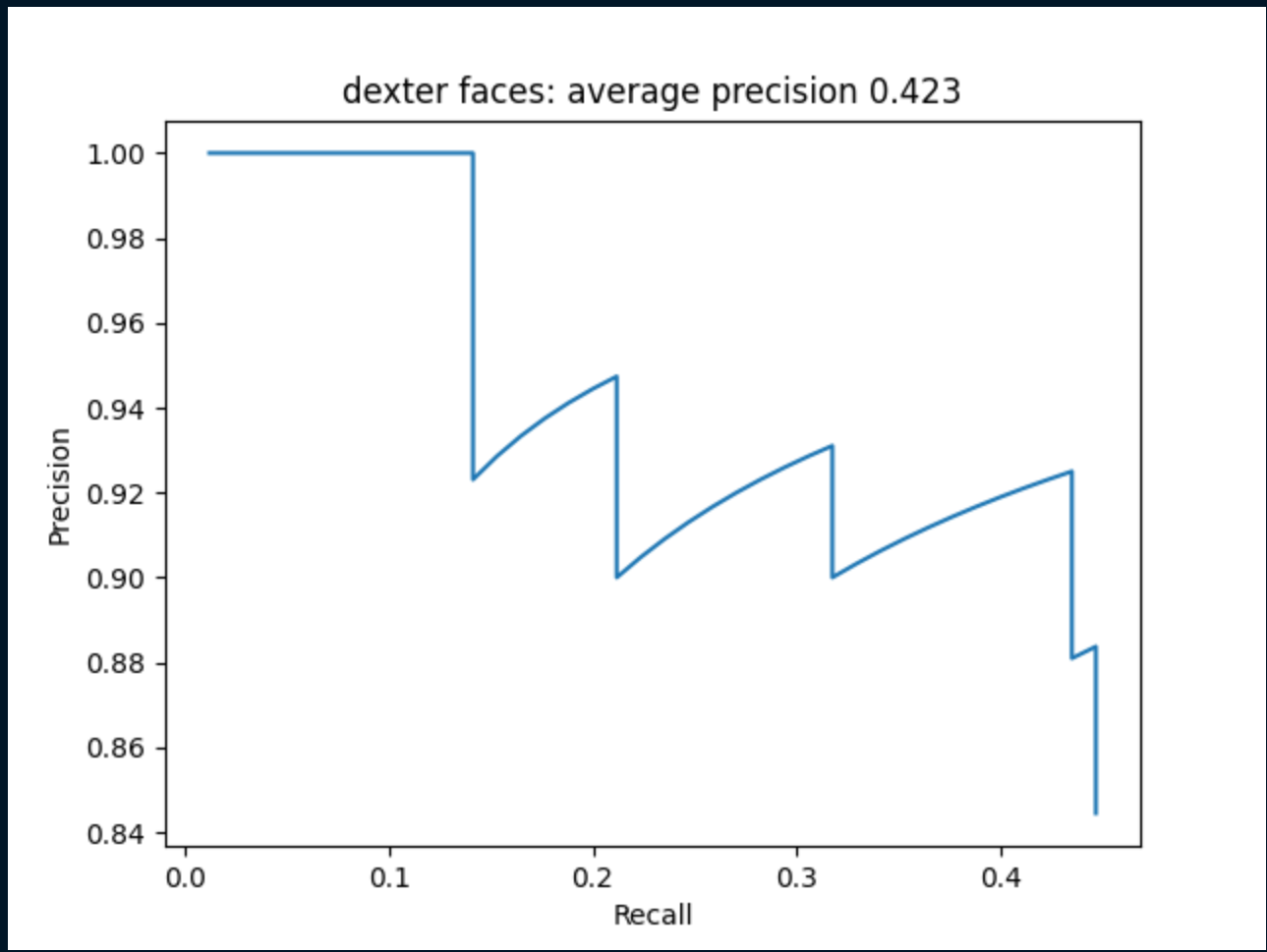
Pentru partea de Task2, am considerat ferestrele deja detectate la Task1, peste care am rulat o retea CNN clasica, in mare parte preluata de la provocarea Kaggle de la IA din anul 2, la care am facut mici modificari. Pentru a organiza mai bine datele de antrenare, am creat in scriptul DatasetCreationPlusBounding un script care creaza in folderul fisiere\_salvate\_algoritm folderul train\_cnn, care pune in fiecare folder cu denumirea claselor, imagini din clasa respectiva, pentru Dexter, DeeDee, Mom, Dad.

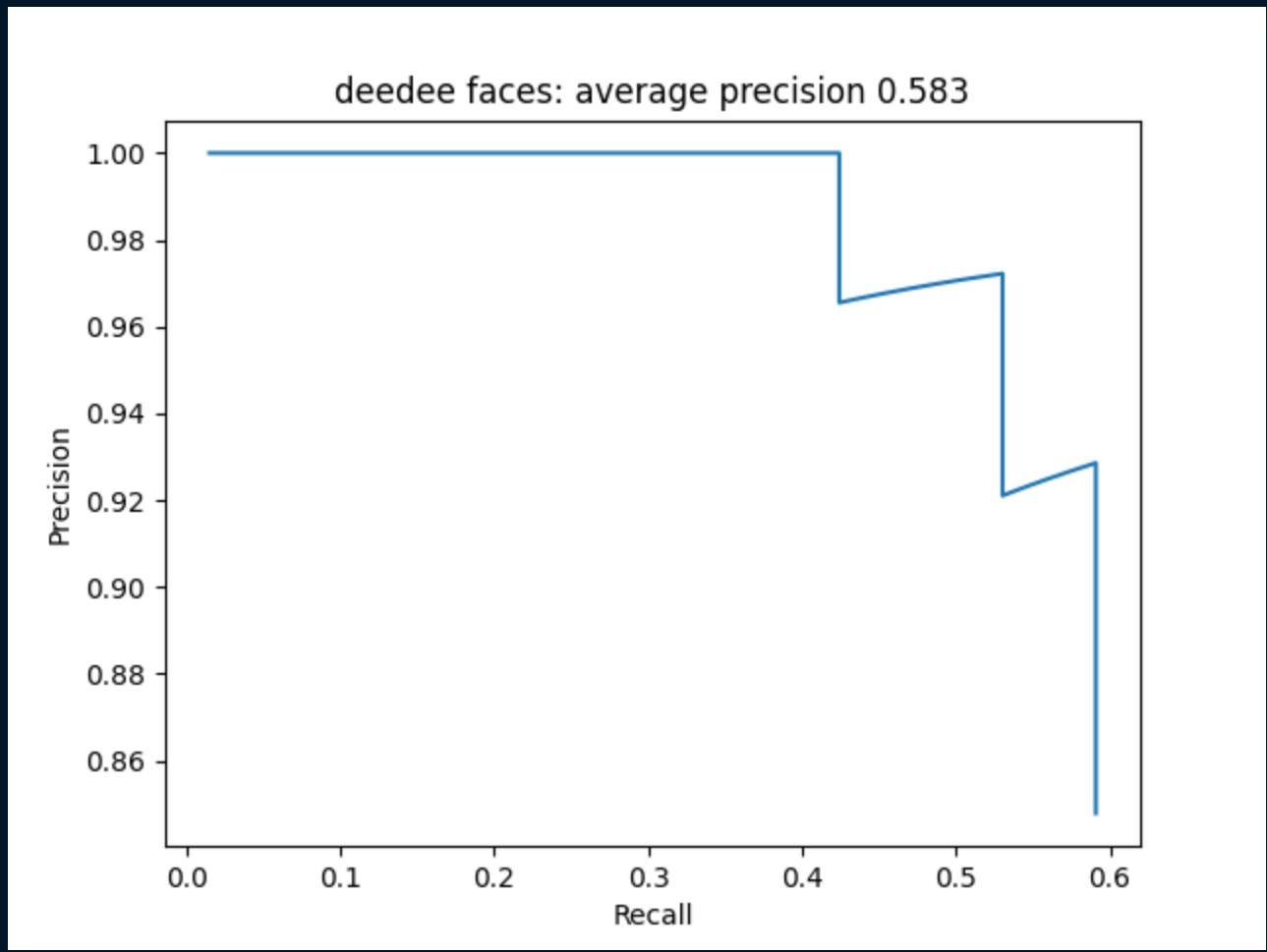
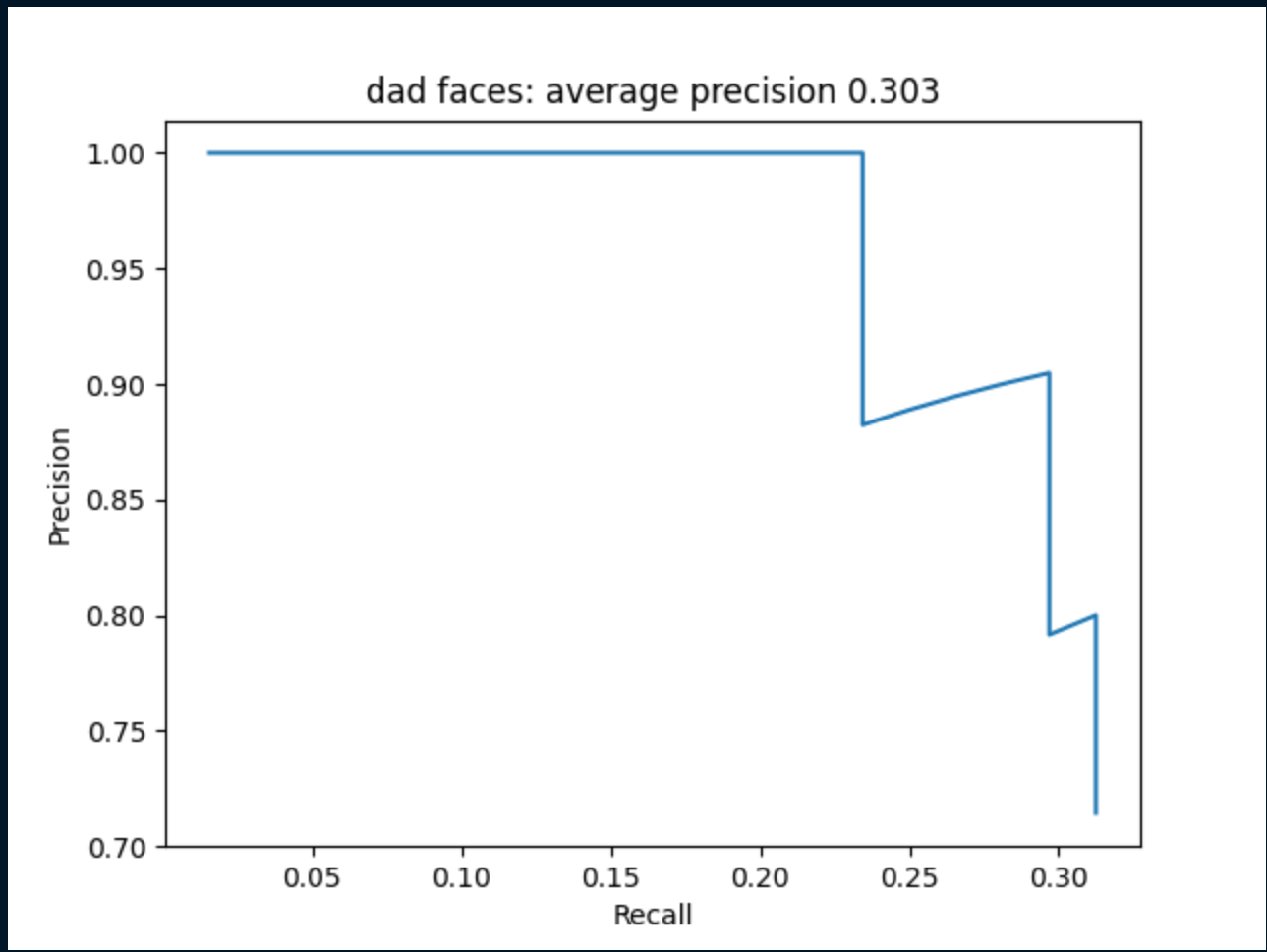
Pentru partea de preprocesare a datelor de antrenare pentru CNN, am ales dimensiunea constanta de 200x200, rotatii random, flip-uri de pana la 15 grade, luminoziate, contrast, saturation random, facem si un gaussian blur cu kernel de 3x3, apoi folosim normalizarea cu valorile din statisticile de la ImageNet.

Modelul folosit este un CNN care contine mai multe straturi convolutionale si straturi fully connected. Sunt 5 blocuri principale, fiecare continand un strat convolutional, o activare ReLu, un strat de batch normalization si un strat de max pooling pentru a reduce dimensionalitatea. In primul bloc, folosesc un strat convolutional cu 3 canale (pentru RGB) care aplica 32 de filtre cu dimensiunea 3x3 si dimensiunea imaginii se pastreaza datorita padding-ului. Batch normalization calculeaza media si varianta fiecarei dimensiuni si normalizeaza valorile. Dupa aplicarea primului filtru se produc 32 de canale pe fiecare imagine, apoi, cu fiecare strat, se creste numarul de canale, iar dimensiunea fiecarui filtru este injumatatita prin tehnica de max pooling. La stratul final al retelei, vectorul de dimensiune 256x12x12 se liniarizeaza, se aplica tehnica de dropout unde jumatate din conexiuni sunt taiate pentru a determina reteaua sa se adapteze la noi feature-uri. Aceasta tehnica previne overfitting-ul. Vectorul de 256 de valori este transformat in vector de 4 valori ce reprezinta probabilitatile de apartenenta la fiecare clasa. Pentru faza de antrenare, am ales criteriul de loss crossentropy care masoara diferenta intre valorile prezise de model si valorile adevarate, iar ca optimizator am ales RMSProp cu un learning rate de 0.001 pentru actualizarea weight-urilor modelului in timpul antrenarii. Apoi, in functia de antrenare, primim modelul, seturile de date, criteriul loss, optimizatorul, numarul de epoci de antrenare si fisierul pentru salvarea modelului. Apoi, pentru fiecare imagine, o mutam pe GPU, resetam gradientul acumulat la iteratiile anterioare, calculam predictiile modelului pentru imaginea curenta si calculam loss-ul, apoi calculam gradientul loss-ului fata de toti parametri modelului si actualizam parametrii modelului folosind gradientul calculat. La final, adaugam loss-ul batch-ului curent la loss-ul total cu ponderea numarului de imagini din batch. Loss-ul mediu va fi asadar tot ce s-a acumulat ca loss curent impartit la numarul de imagini din tot dataset-ul.

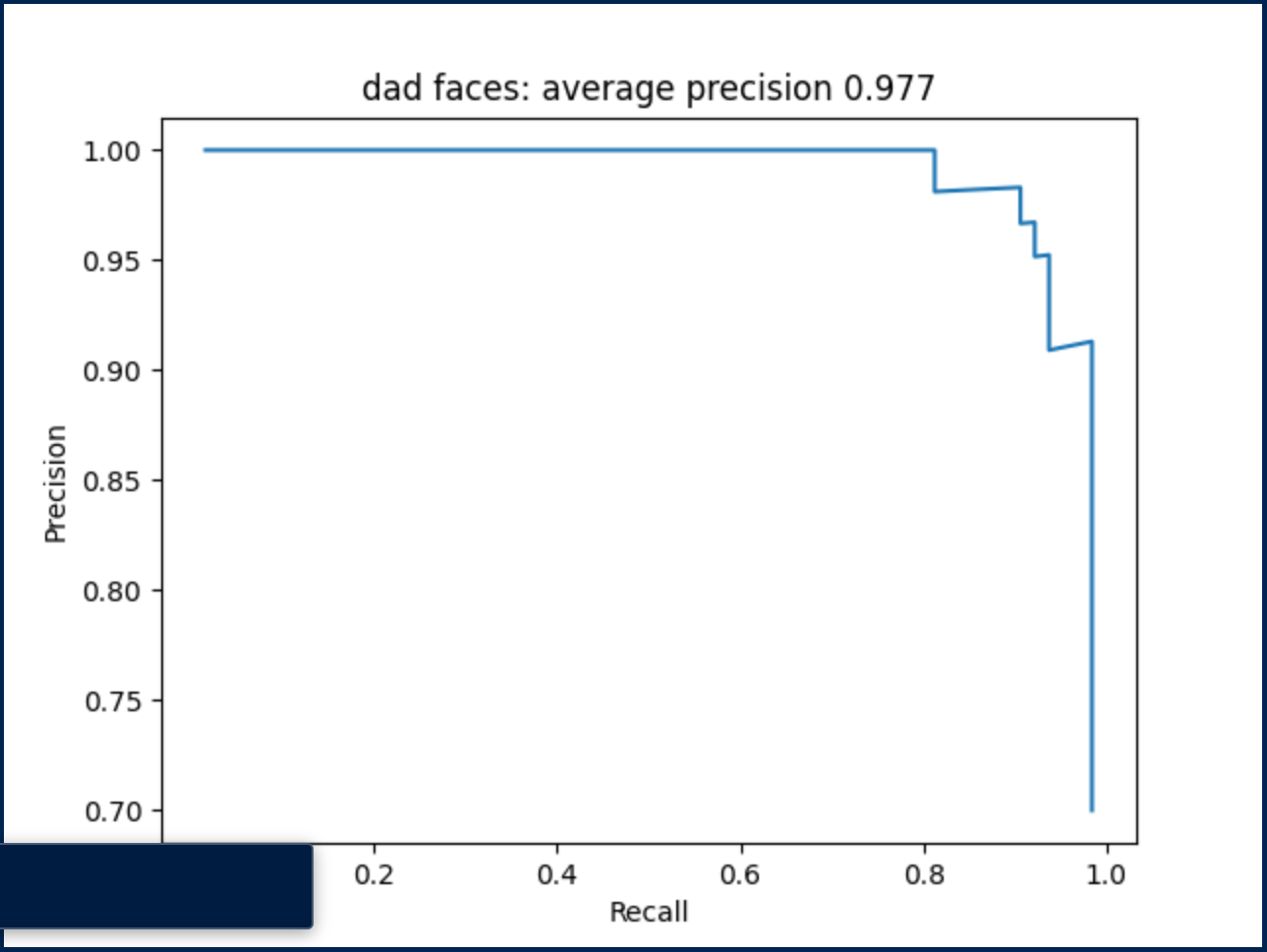
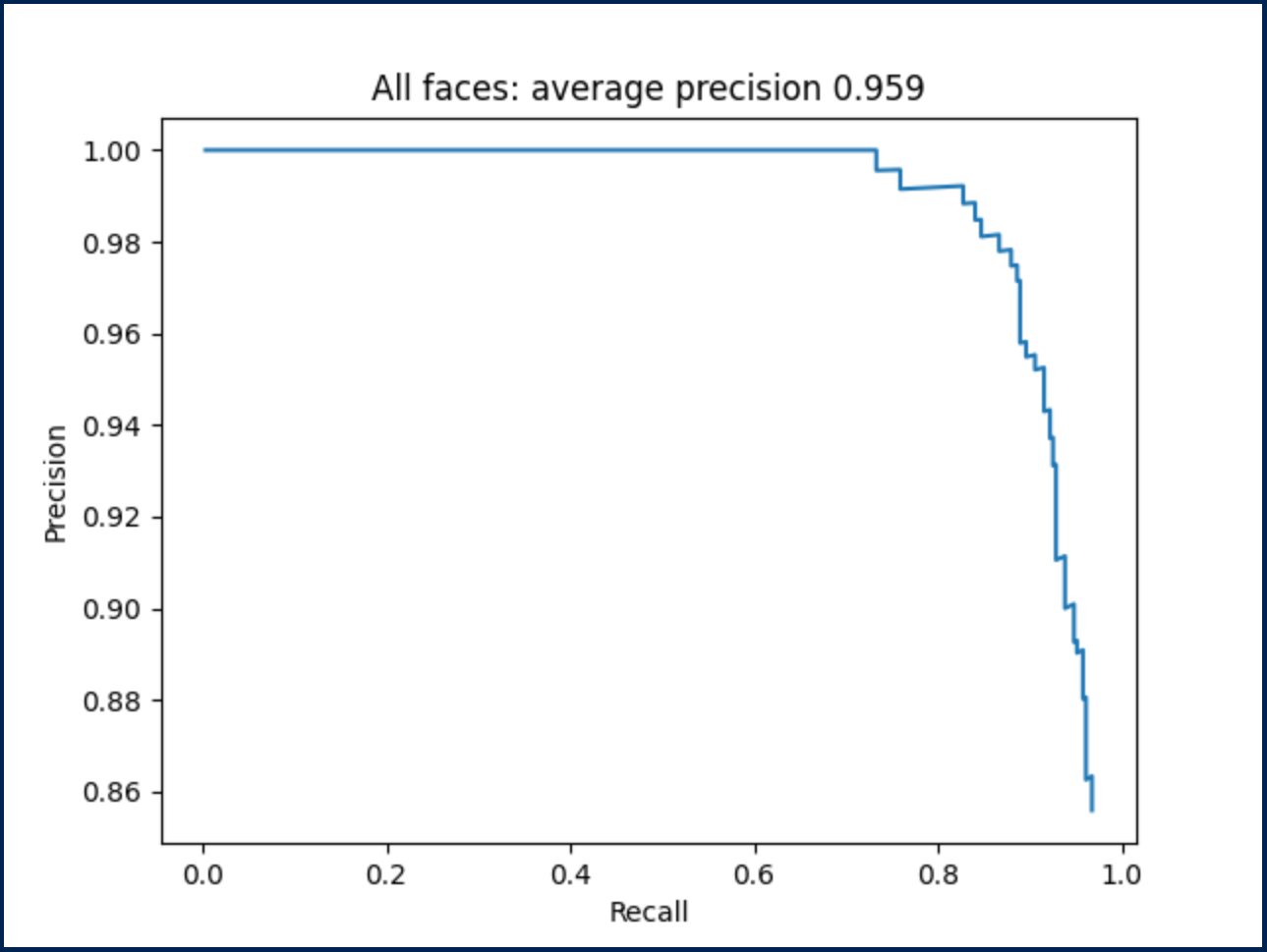
In faza de validare, setam modelul in modul de evaluare si dezactivam calculul gradientului, iar pentru fiecare imagine ii comparam label-ul prezis cu cel adevarat si calculam acuratetea la epoca curenta. Daca acuratetea este cea mai buna de pana acum, salvam modelul.

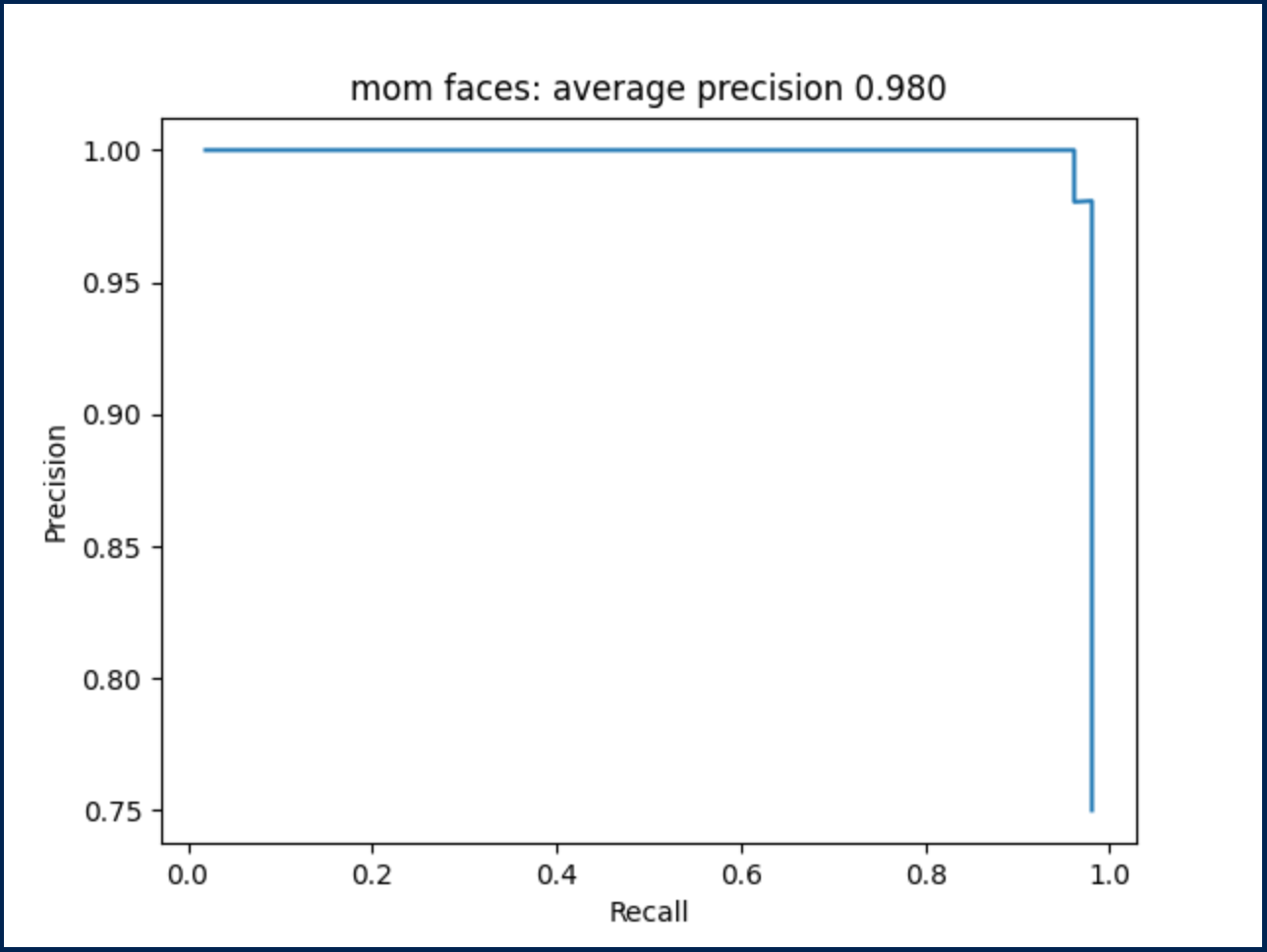
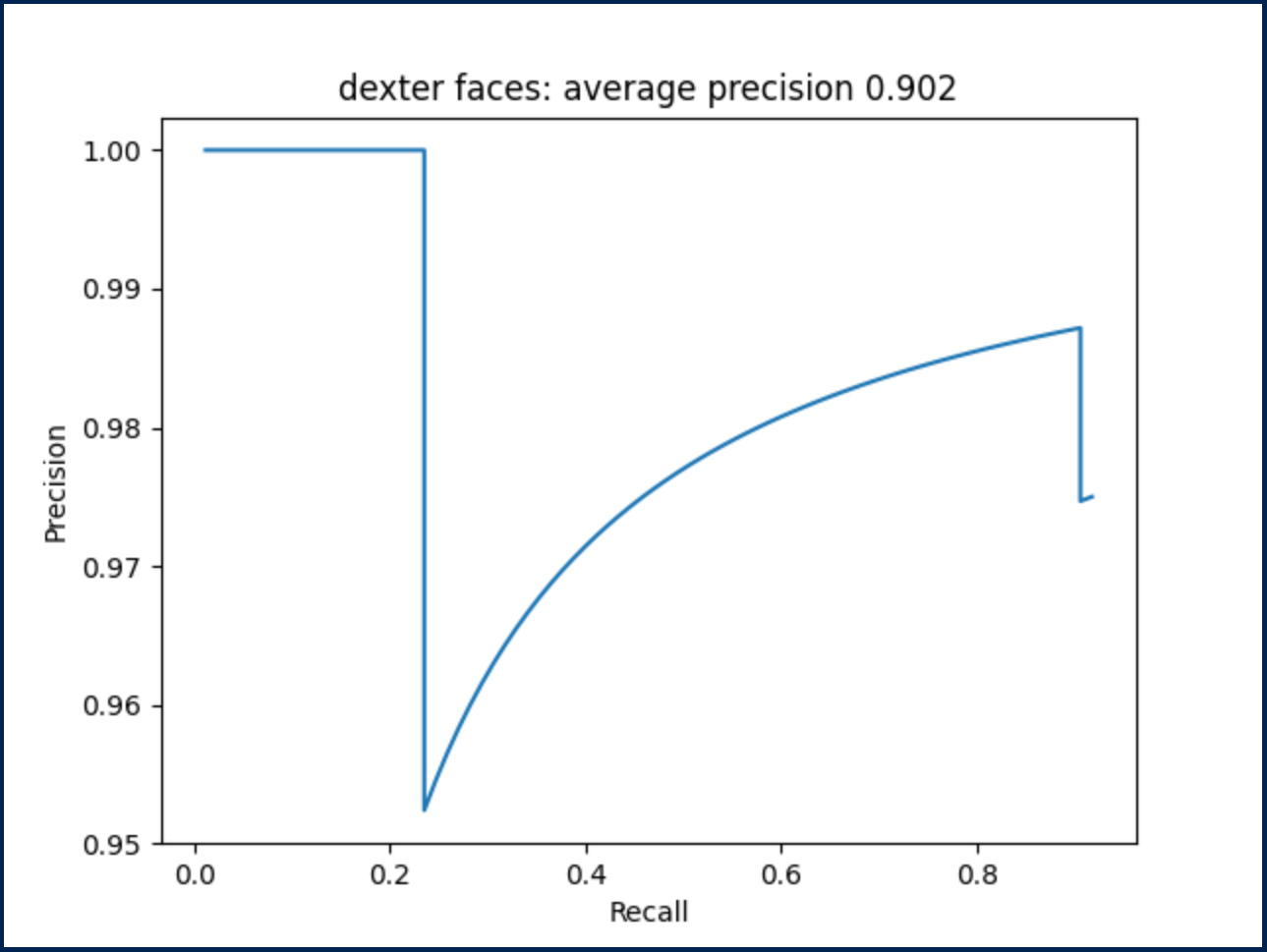
Rularea modelului de CNN are loc pentru fiecare fereastra detectata la task-ul 1, obtinand urmatoarele precizii:

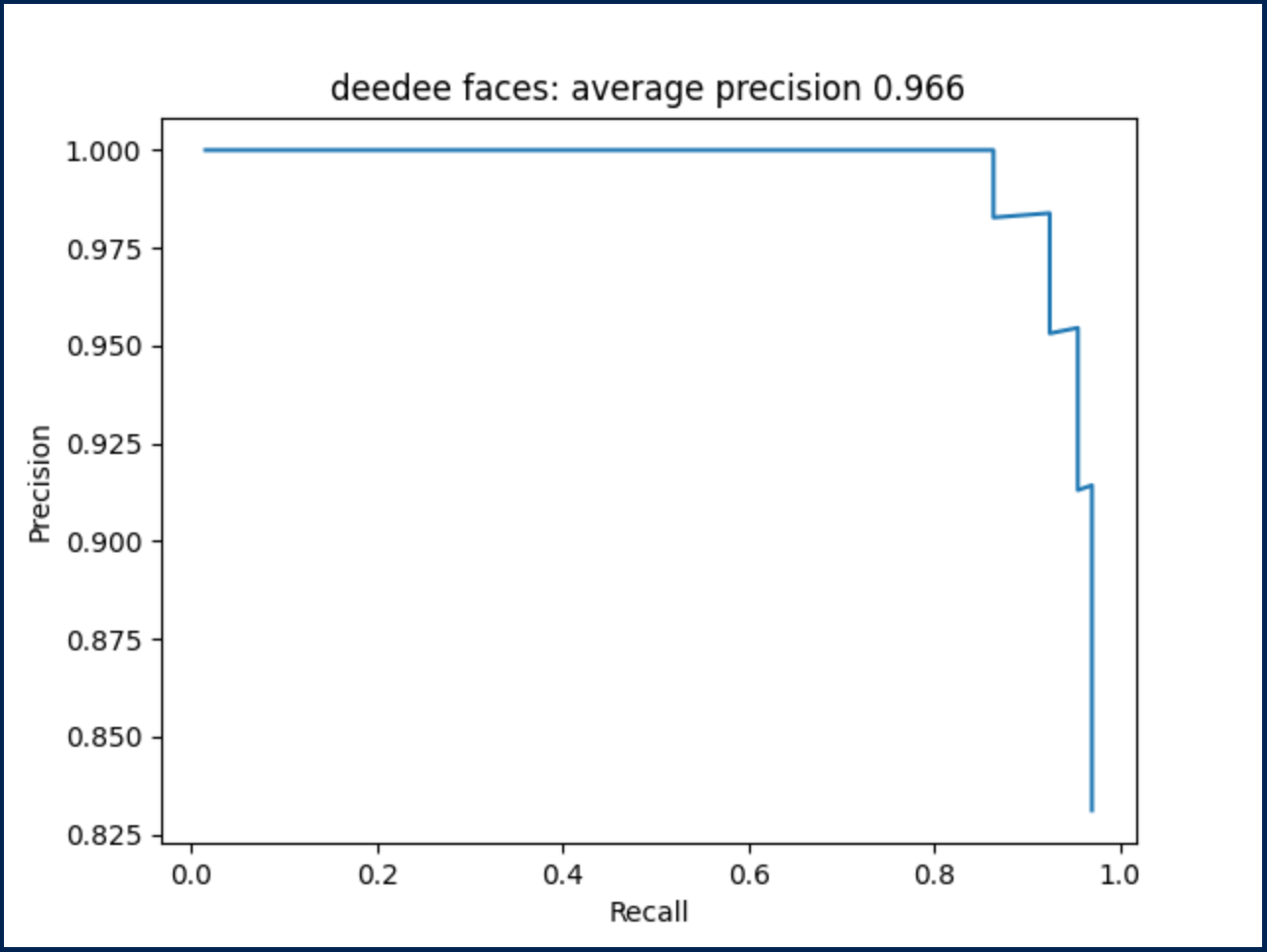




Ca optiune mai buna, cu precizii mai adecvate, am construit si un model YOLO, care preia imaginile de antrenare ca foldere cu denumirile echivalente cu label-uri pentru cele 4 clase. In fisierul PrepareData.py, se creaza aceste foldere, plus se face setup-ul pentru modelul YOLOv8. In folderul dataset se creaza folderele images si labels, unde, pentru fiecare imagine, avem clasa asociata si pozitia in imaginea respectiva, asadar modelul va cunoaste pozitiile fetelor din datele de antrenare pe fiecare imagine. Modelul are nevoie doar de aceasta configurare, el este descarcat, se dau numarul de epoci (50), dimensiunea standard a imaginilor din dataset este setata la 512, apoi se ruleaza fisierul TrainYOLO, care seteaza folderele si face setup-ul pentru structura, apoi antreneaza modelul pe toate imaginile impreuna cu clasele acestora. Apoi, rulam pe dataset-ul de validare. Pentru fiecare imagine data ca parametru modelului, avem returnate automat de model pozitiile fetelor plus clasificarea acestora, nu mai sunt necesare preprocesari suplimentare, sliding windows, antrenari, descriptori, etc. Pe dataset-ul de validare, cu modelul YOLOv8, am obtinut urmatoarele precizii:







Proiectul a avut ca scop detectarea si recunoasterea fetelor personajelor din desenul Dexter, abordand problema in doua etape principale. Prima etapa a implicat detectarea fetelor utilizand descriptori HOG pentru extragerea caracteristicilor si un clasificator SGD cu invatare incrementala, aplicat prin tehnica sliding window. Aceasta abordare a obtinut o precizie de 0.352 pe datele de validare. In a doua etapa, pentru clasificarea personajelor, s-au testat o retea neuronala convolutiva (CNN) cu 5 straturi, iar, in final, un model YOLOv8. Modelul YOLO s-a dovedit mai eficient, combinand detectia si clasificarea intr-un singur pas. Principalele provocari au fost gestionarea resurselor, echilibrarea exemplelor pozitive si negative si ajustarea parametrilor. Pentru imbunatatirea rezultatelor s-au aplicat augmentarea datelor, paralelizarea proceselor si normalizarea. Modelul YOLO a oferit cele mai bune rezultate, cu preprocesare minima si performanta mare.