CAVA - Tema 2
Detectarea și recunoașterea facială în Laboratorul lui Dexter

Chiruş Mina-Sebastian

Ianuarie 2025

1 Introducere

Această lucrare descrie soluția propusă pentru detectarea și recunoașterea facială a personajelor din serialul de animație *Laboratorul lui Dexter*.

Detectarea fețelor a fost realizată folosind descriptori extrași dintr-o rețea convoluțională bine cunoscută, AlexNet, pre-antrenată pe un set de date general, și o rețea neuronală simplă antrenată pe acești descriptori pentru a distinge între exemplele pozitive (fețe) și cele negative (non-fețe) date de o fereastră glisantă.

Pentru recunoașterea facială, soluția utilizează o arhitectură neuronală similară, extinsă pentru clasificarea multi-clasă, care permite identificarea fiecăruia dintre cele cinci personaje țintă.

Soluția inițială a fost bazată pe descriptori HOG cu un clasificator SVM dar acuratețea nu a fost una satisfăcătoare.

De asemenea, a fost utilizat un model YOLO, antrenat pe setul de date furnizat la laborator, care a demonstrat o performanță excelentă, atingând rezultate aproape perfecte în detectarea fețelor.

2 Arhitectura Soluției

2.1 Descriere Generală

Codul este structurat modular, fiecare componentă având un rol bine definit:

- **Detecție Facială**: Ferestrele glisante trec peste variante mărite și micșorate ale imaginii de bază(metoda piramidei). Pentru fiecare fereastră extragem descriptorii din rețeaua AlexNet și punem clasificatorul să facă o predicție.
- **Recunoaștere Facială**: Clasificarea fețelor detectate în cinci clase corespunzătoare personajelor Dexter, DeeDee, Mom, Dad și Unknown, folosind un clasificator asemănător cu cel al detecției faciale.

- **Metode Complementare**: Integrarea unui model YOLO pentru detecție rapidă și eficientă.
- **Generarea Exemplelor**: Crearea de seturi de date pozitive și negative pentru antrenarea și validarea rețelelor.

2.2 Structura Codului

- **FaceDetectinNN**: Clasa principală care implementează întreaga metodologie, de la generarea exemplelor până la detectarea și recunoașterea facială.
- **DescriptorNNMultiClass**: O rețea neuronală folosită pentru clasificarea descriptorilor în clasele personajelor.
- **DescriptorsNN**: Rețea neuronală pentru clasificarea binară a descriptorilor (față sau non-față).
- **Tools**: Funcții auxiliare, precum *sliding window*, *non-maximal suppression* și calculul *intersection over union*.
- **ImagesLoader**: Clasa responsabilă de încărcarea imaginilor și extragerea de patch-uri bazate pe adnotări.
- **AnnotationsParser**: Parsarea fișierelor de adnotări pentru imagini.
- **Utilizarea YOLO**: Integrarea YOLO pentru detecția fețelor folosind modele re-antrenate.

3 Implementare

3.1 Generarea Exemplelor

Exemplele pozitive nu au necesitat o prelucrare suplimentară semnificativă. Am dezvoltat o clasă specială, **AnnotationsParser**, care are rolul de a încărca fișierele de adnotări (*.txt*) furnizate, asociate fiecărei imagini din setul de date. Această clasă parcurge fișierele, le procesează și returnează un dicționar de forma:

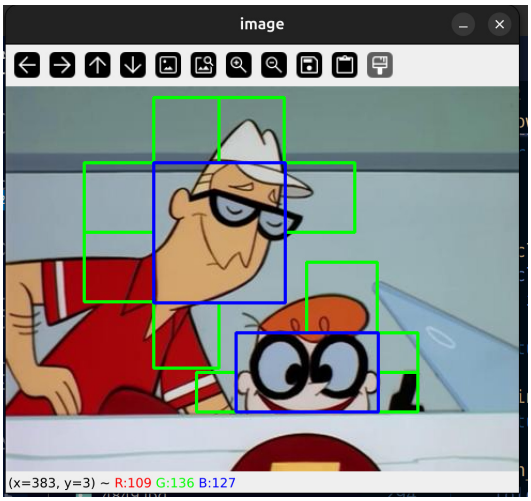
`personaj: listă anotări`.

Pentru fiecare personaj (*Dexter, DeeDee, Mom, Dad, Unknown*), adnotările constau în coordonatele ferestrelor delimitatoare (*bounding boxes*) care specifică poziția fețelor în imagini.

Exemplele negative, pe de altă parte, au necesitat mai multe iterații și ajustări pentru a obține o selecție eficientă.

Abordarea inițială a implicat utilizarea adnotărilor pentru a genera ferestre negative adiacente fiecărei fețe. Pentru fiecare dreptunghi delimitator

(*bounding box*) asociat unei fețe, au fost extrase ferestre de dimensiune dinamică d din vecinătatea imediată care să nu se interseteze cu o adnotare. Scopul acestei strategii a fost de a învăța algoritmul să acorde un scor scăzut secțiunilor de lângă personaj (precum gâtul, umărul sau alte regiuni din apropiere), astfel încât să detecteze doar fața propriu-zisă.



Exemplele negative generate au fost analizate și sortate în funcție de complexitatea lor, folosind măsuri de entropie a histogramei de culoare și densitatea marginilor. Acest proces a avut ca scop prioritizarea imaginilor complexe, care ar putea contribui mai semnificativ la antrenarea modelului. Metodologia este detaliată mai jos.

Calculul entropiei histogramei de culoare: Pentru fiecare imagine, histograma de intensitate a fost calculată, iar entropia acesteia a fost utilizată pentru a măsura diversitatea pixelilor. Formula utilizată este:

$$H = -\sum_i p_i \cdot \log_2(p_i + \epsilon),$$

unde p_i reprezintă probabilitatea asociată fiecărui nivel de intensitate, iar ϵ este o valoare mică adăugată pentru stabilitate numerică.

Densitatea marginilor: Marginile din imagine au fost extrase folosind detectorul de margini Canny, iar densitatea acestora a fost calculată ca raportul dintre numărul de pixeli ce aparțin marginilor și numărul total de pixeli din imagine:

$$D = \frac{\text{Număr pixeli margine}}{\text{Număr total pixeli}}.$$

Algoritm de sortare: Funcția `process_images` procesează fiecare imagine dintr-un director specificat, calculează entropia și densitatea marginilor, și sortează imaginile în două categorii:

- **Imagini complexe**: Imagini cu entropie ridicată ($H >$ prag) sau densitate mare a marginilor ($D >$ prag).
- **Imagini cu complexitate scăzută**: Restul imaginilor care nu îndeplinesc aceste criterii.

Rezultatul nu a fost pe măsura așteptărilor. Deși metoda a permis crearea unui set diversificat de exemple negative, algoritmul a întâmpinat dificultăți în separarea clară a fețelor personajelor de alte regiuni similare plus că a avut o problemă fundamentală: Ferestrele generate erau exclusiv pătrate, deși algoritmul a fost antrenat și cu imagini dreptunghiulare, redimensionate la 224x224 pixeli pentru pipeline-ul din AlexNet.

Abordarea Finală

Pentru a îmbunătăți procesul de generare a exemplelor negative și a reduce discrepanțele dintre ferestrele pozitive și negative, am împărțit fiecare imagine de antrenare în ferestre de dimensiuni egale cu cele utilizate în metoda de evaluare: $(80, 80)$, $(80, 48)$, $(48, 80)$.

Procesul detaliat este următorul:

- **Generarea ferestrelor**: Fiecare imagine a fost împărțită în ferestre de dimensiuni predefinite, corespunzătoare aspectului ferestrelor utilizate pentru detectare.
- **Calculul intersecției cu adnotările**: Pentru fiecare fereastră generată, a fost măsurată aria de intersecție cu dreptunghiurile delimitatoare (*bounding boxes*) din adnotările imaginii.
- **Filtrarea ferestrelor**: Ferestrele care aveau o valoare $IoU < 0.3$ au fost selectate pentru a evita suprapunerea excesivă cu fețele personajelor, dar în același timp pentru a include regiuni similare.
- **Sortarea ferestrelor**: Ferestrele negative au fost sortate descrescător după aria de intersecție. Această strategie a permis includerea unor fragmente din fețele personajelor, dar și pe cele care nu le cuprindeau.

În final am iterat prin multiple detecții ale setului de antrenare pentru a găsi exemple negative puternice(hard negative mining). La fiecare iterație dura din ce în ce mai mult să găsesc exemple negative, ceea ce a indicat că metoda este bună.

3.2 Generarea Descriptorilor

Pentru detectarea și recunoașterea facială, am folosit descriptorii dați de rețeaua convoluțională **AlexNet**, metodă prezentată la Laboratorul 11.

Procesul de generare a descriptorilor include următoarele etape:

- **Preprocesarea imaginilor**: Fiecare imagine sau patch extras este redimensionat la dimensiunea standard (224×224) și normalizat conform parametrilor specifici AlexNet (*mean* și *std*).
- **Extragerea descriptorilor**: Imaginile preprocesate sunt procesate prin rețeaua AlexNet, iar descriptorii sunt extrași din ultimul strat convoluțional.

```
def get_descriptor(image, model, device):
    preprocess = transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225]),
    ])
    image_tensor = preprocess(image).unsqueeze(0).to(device)
    model.eval()
    with torch.no_grad():
        features = torch.zeros(256)  # Dimensiunea
            descriptorului final
        def hook_fn(module, input, output):
            nonlocal features
            features.copy_(output.mean([2, 3]).squeeze())
        handle = model.features[-1].register_forward_hook(
            hook_fn)
        model(image_tensor)
        handle.remove()
    return features.cpu().numpy()
```

Listing 1: Generarea descriptorilor cu AlexNet

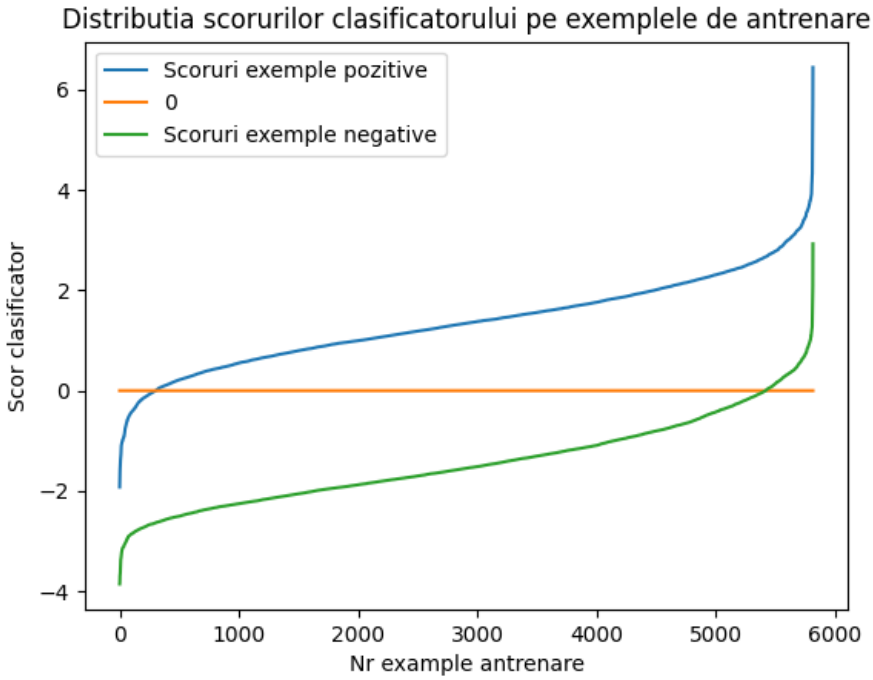
Descriptorii generați sunt utilizați în două etape principale:

- **Detecția facială**: Diferențierea între exemplele pozitive (fețe) și negative (non-fețe), folosind clasificatori binari.
- **Recunoașterea facială**: Clasificarea multi-clasă a descriptorilor pentru a identifica personajele individuale (Dexter, DeeDee, Mom, Dad, Unknown).

Un dezavantaj al acestei metode este că descriptorii trebuie generați pentru fiecare fereastră glisantă, față de descriptorii HOG care se puteau genera pentru toată imaginea și se decupau pe coordonatele ferestrei glisante.

3.3 Detecția facială

Am ales să construiesc o rețea neuronală și să nu folosesc un SVM deoarece SVM-ul, la fiecare nouă iterație de exemple negative puternice, devenea mai instabil:



Așadar, am construit o rețea simplă fully-connected, cu o singură ieșire care reprezintă probabilitatea ca descriptorii introduși să fie ai unei fețe, adăugând pierderea de tip BCELoss pentru a încuraja" modelul să dea valori mai apropiate de 0 sau 1.

```
nn.Sequential(
            nn.Linear(self.embedding_dim, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, 1),
            nn.Sigmoid()
        )
```

Listing 2: Rețea de clasificare binară

3.4 Recunoașterea Facială

Pentru recunoașterea fețelor detectate, am folosit o arhitectură asemănătoare, singurele lucruri schimbate au fost schimbarea funcției de pierdere BCELoss cu CrossEntropyLoss, numărul de ieșiri ale rețelei este egal cu numărul claselor pe care trebuie să le detecteze(5) și renunțarea la Sigmoidă deoarece se aplică Softmax pentru clasele multiple.

```
nn.Sequential(
            nn.Linear(embedding_dim, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, num_classes)
        )
```

Listing 3: Rețea de clasificare a personajelor

Pentru clasa Unknown, pozele au trebuit augumentate deoarece aveam un număr de doar 300 de fețe, pe când la celelalte clase aveam un număr de minim 1033 fețe.

```
for ann in tqdm(personage_annots, desc=f"Processing {personage}
    examples"):
                    img = cv.imread(ann.get_file())
                    img = img[ann.bbox[1]:ann.bbox[3], ann.bbox
                        [0]:ann.bbox[2]]
                    img = cv.resize(img, (224, 224))
                    descriptor = self.get_descriptor(img)
                    personage_descriptors.append(descriptor)
                    if personage == 'unknown':
                        img_aug = np.fliplr(img)
                        descriptor = self.get_descriptor(
                            img_aug)
                        personage_descriptors.append(descriptor
                            )
                        for i in range(1, 2):
                            img_rot = cv.rotate(img, cv.
                                ROTATE_180)
                            descriptor = self.get_descriptor(
                                img_rot)
                            personage_descriptors.append(
                                descriptor)
                personages_descriptors[personage] = np.array(
                    personage_descriptors)
                np.save(personage_file, personages_descriptors[
                    personage])
```

Listing 4: Multiplicare exemple pentru datele de la clasa Unknown

Am încercat să suplimentez valorile lipsă de la unknown cu descriptori negativi dar rezultatul a fost substanțial mai slab față de varianta cu imagine rotie.

3.5 Utilizarea YOLO

Pentru a antrena modelul YOLO pe setul de date dat, am construit o funcție care folosește clasa **AnnotationsParser** pentru a converti toate adnotările in formatul YOLO pentru antrenare:

```
def export_to_yolo(ann_parser, output_dir="yolo_dataset"):
    label_map = {
        'mom': 0,
        'dad': 1,
        'dexter': 2,
        'deedee': 3,
        'unknown': 4
    }

    file_to_anns = {}
    for lbl, ann_list in ann_parser.annotations.items():
        for ann in ann_list:
            img_path = ann.get_file()
            file_to_anns.setdefault(img_path, []).append(ann)

    all_files = list(file_to_anns.keys())
    random.shuffle(all_files)

    train_count = int(len(all_files) * 0.8)
    train_files = all_files[:train_count]
    val_files = all_files[train_count:]

    img_train_dir = os.path.join(output_dir, 'images', 'train')
    img_val_dir   = os.path.join(output_dir, 'images', 'val')
    lbl_train_dir = os.path.join(output_dir, 'labels', 'train')
    lbl_val_dir   = os.path.join(output_dir, 'labels', 'val')
    for d in [img_train_dir, img_val_dir, lbl_train_dir,
        lbl_val_dir]:
        os.makedirs(d, exist_ok=True)

    def write_to_yolo(files_list, images_dir, labels_dir):
        for img_path in files_list:
            img = cv2.imread(img_path)
            if img is None:
                continue

            if 'dad' in img_path:
                label = 'dad'
            elif 'deedee' in img_path:
                label = 'deedee'
            elif 'dexter' in img_path:
                label = 'dexter'
            elif 'mom' in img_path:
                label = 'mom'
            else:
                label = 'unknown'
            h, w = img.shape[:2]
            base_name       = label + '_' + os.path.basename(
                img_path)
            base_name_noext = os.path.splitext(base_name)[0]
            out_img_path    = os.path.join(images_dir,
                base_name)
```

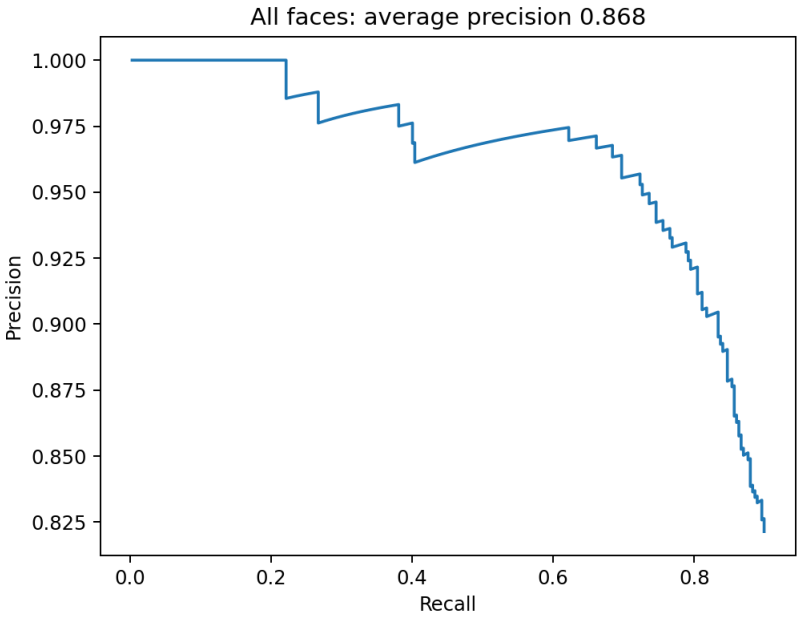
```
            cv2.imwrite(out_img_path, img)

            label_file_path = os.path.join(labels_dir,
                base_name_noext + '.txt')
            with open(label_file_path, 'w') as f:
                for ann in file_to_anns[img_path]:
                    x1, y1, x2, y2 = ann.bbox
                    center_x = ((x1 + x2) / 2) / w
                    center_y = ((y1 + y2) / 2) / h
                    box_w    = (x2 - x1) / w
                    box_h    = (y2 - y1) / h

                    class_id = label_map.get(ann.label,
                        label_map['unknown'])

                    f.write(f"{class_id} {center_x} {center_y}
                        {box_w} {box_h}\n")

    write_to_yolo(train_files, img_train_dir, lbl_train_dir)
    write_to_yolo(val_files,   img_val_dir,   lbl_val_dir)

    print(f"{len(train_files)} train, {len(val_files)} val.")
```

Listing 5: Generare date train si validare pentru YOLO

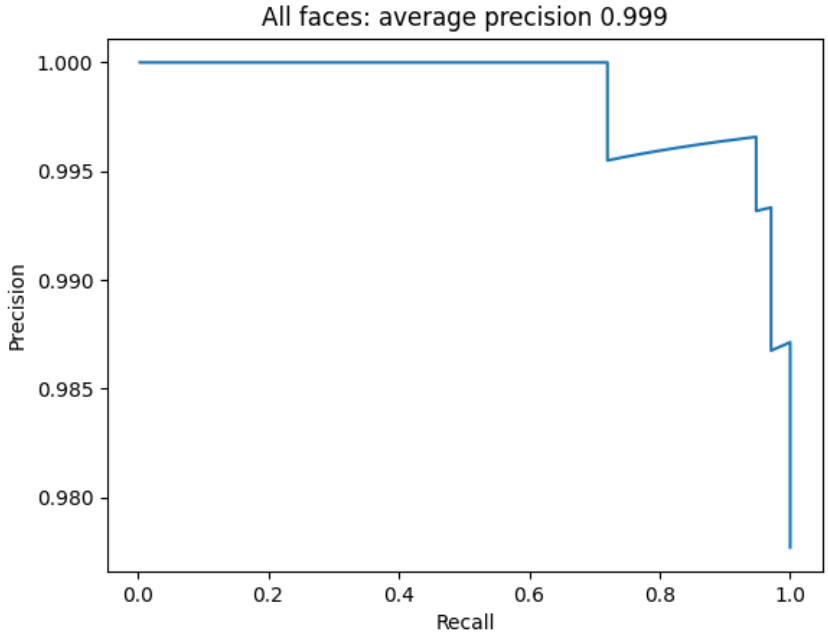
4 Rezultate și Performanță

4.1 Task 1: Detectare Facială

Metoda bazată pe descriptorii AlexNet cu rețeaua neuronală de discriminare a atins o precizie medie (AP) de 86,8%, în timp ce YOLO a obținut un AP de 99,9% pe imaginile de validare.



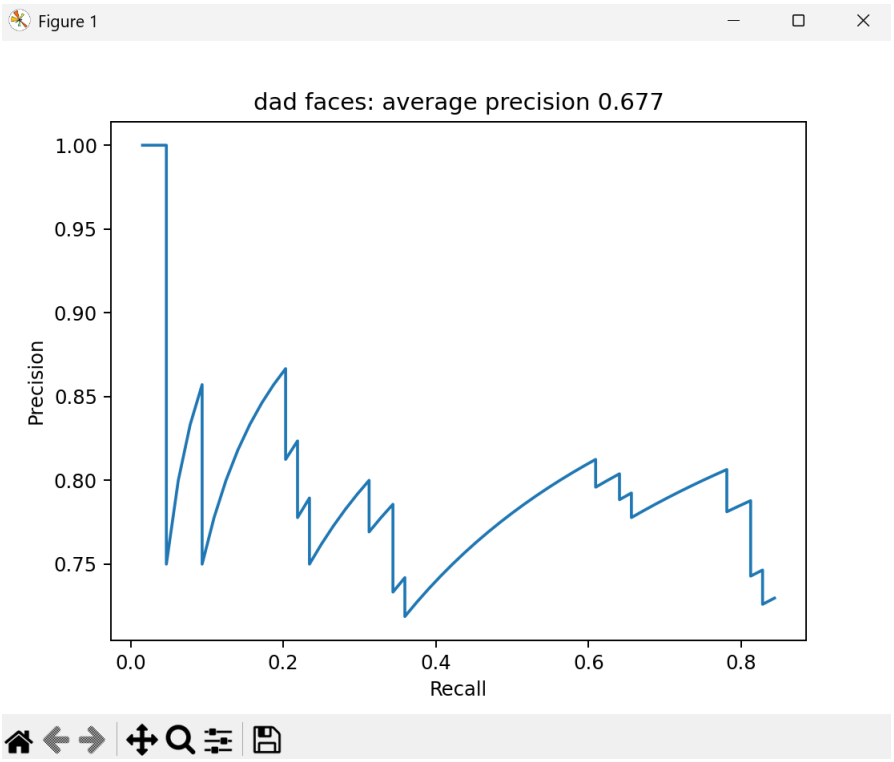
Soluția de bază



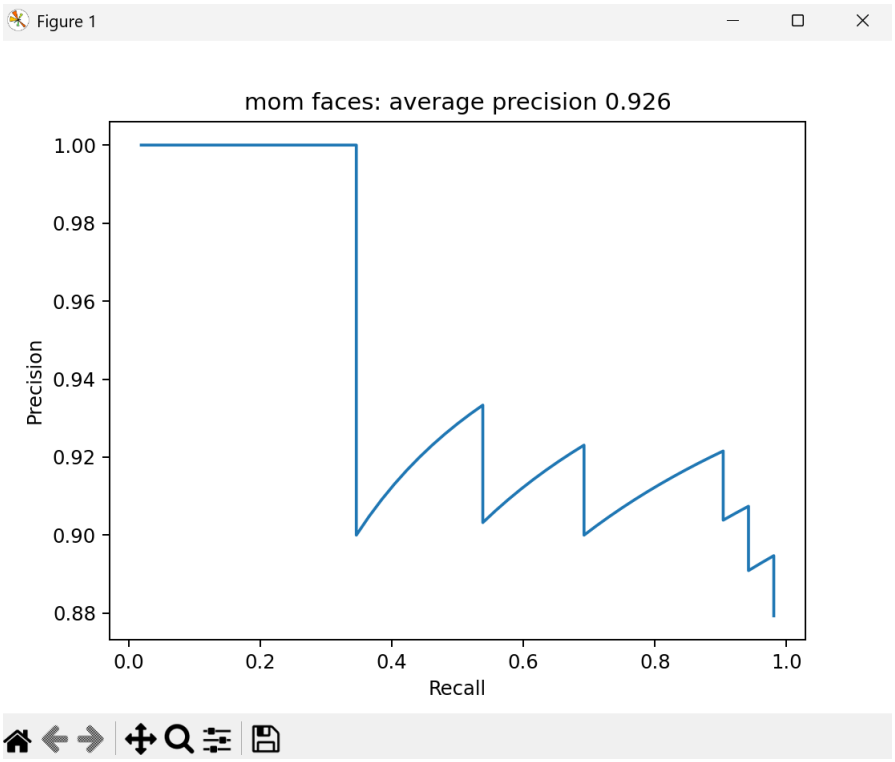
Soluția folosind YOLO

4.2 Task 2: Recunoaștere Facială

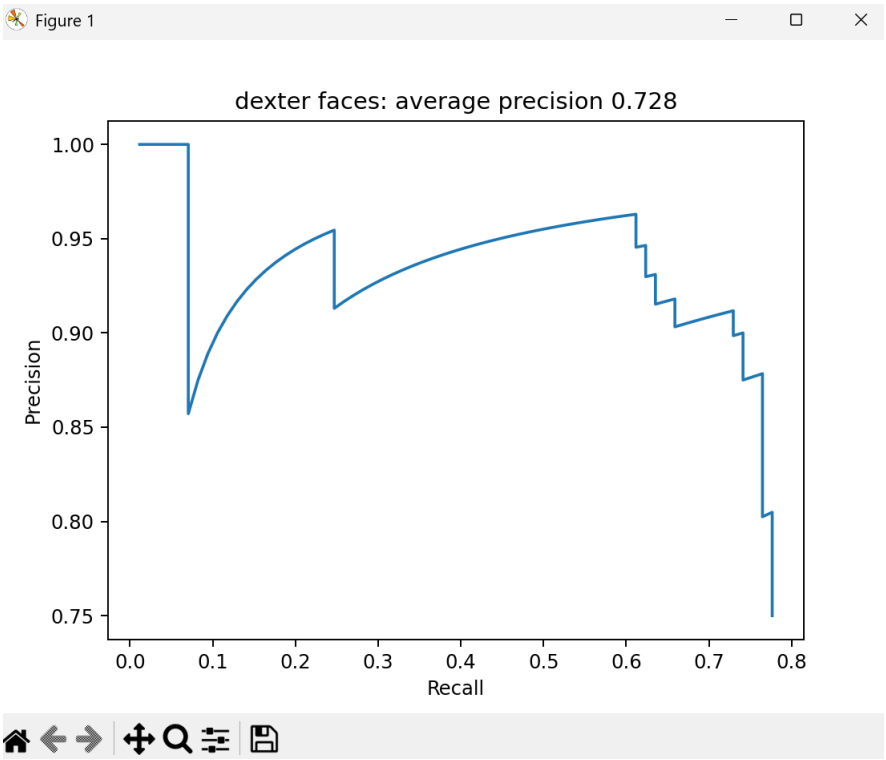
Pentru clasificarea fețelor, rețeaua neuronală multi-clasă a obținut o acuratețe generală de 78,55%, cu rezultate bune pentru toate cele cinci clase de personaje. Modelul YOLO a obținut 99,05% acuratețe medie.



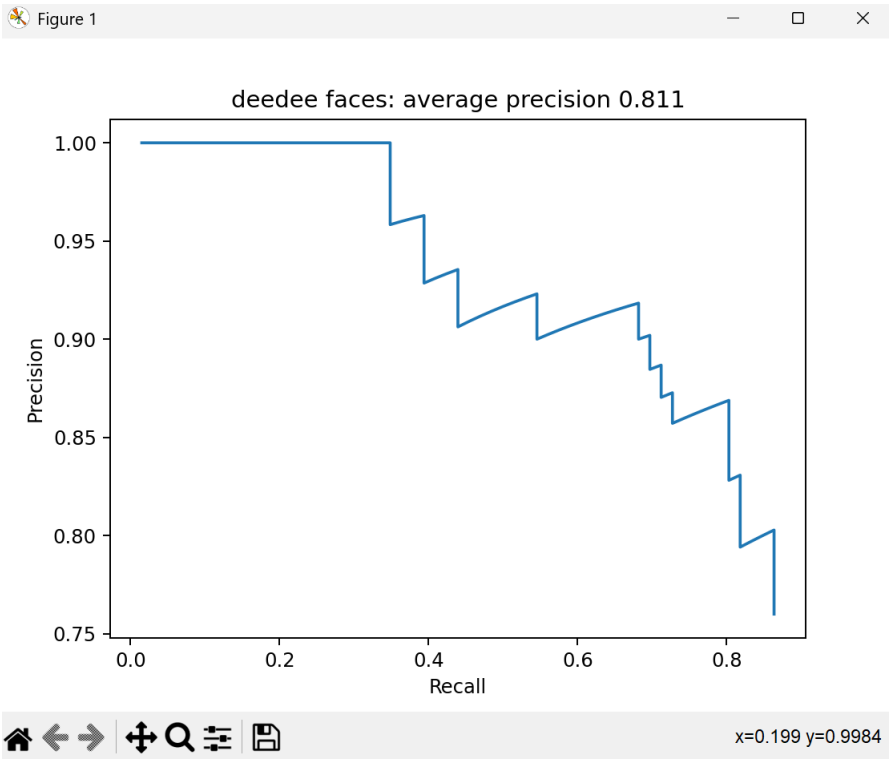
Soluția de bază DAD



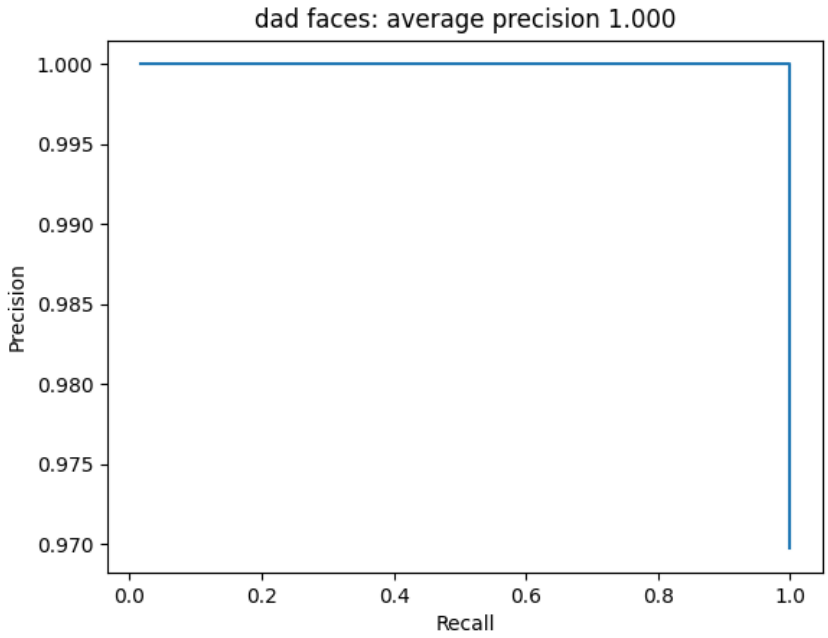
Soluția de bază MOM



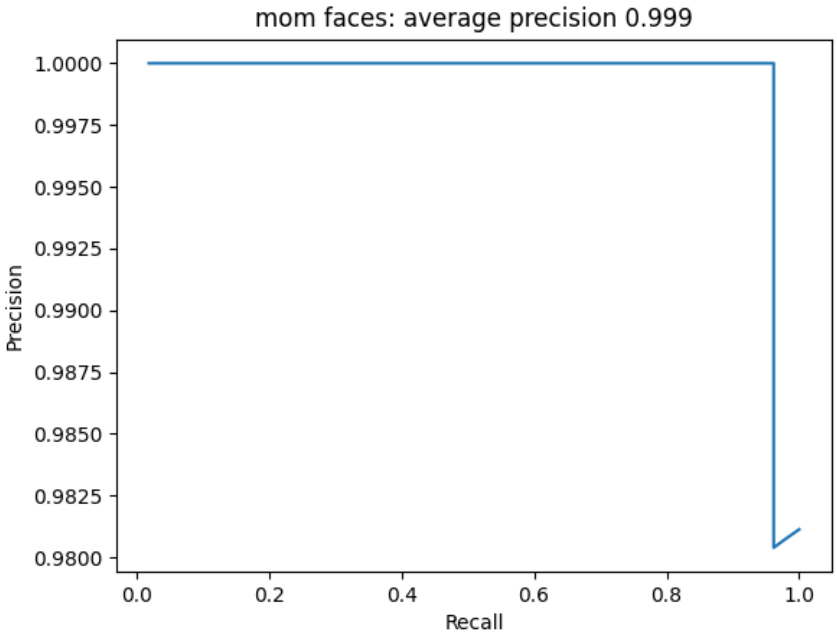
Soluția de bază DEXTER



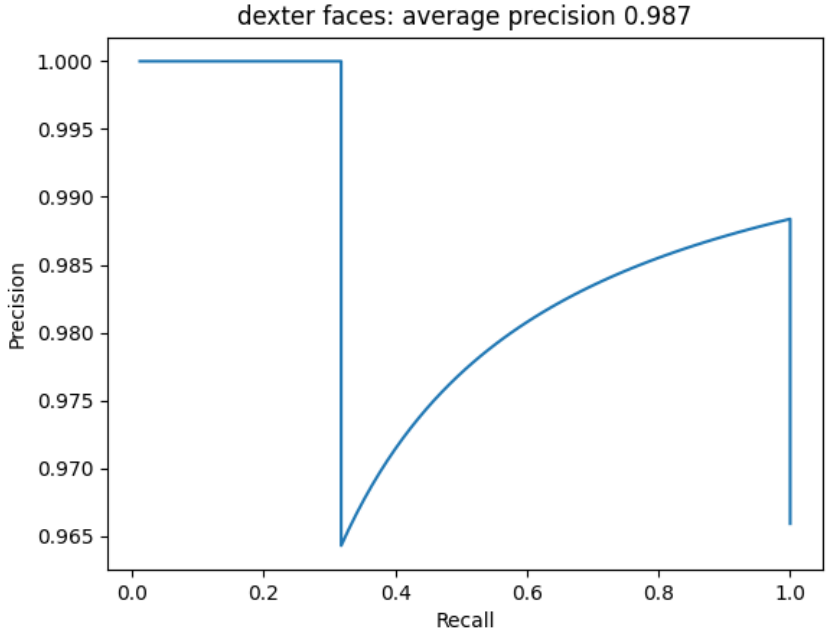
Soluția de bază DEEDEE



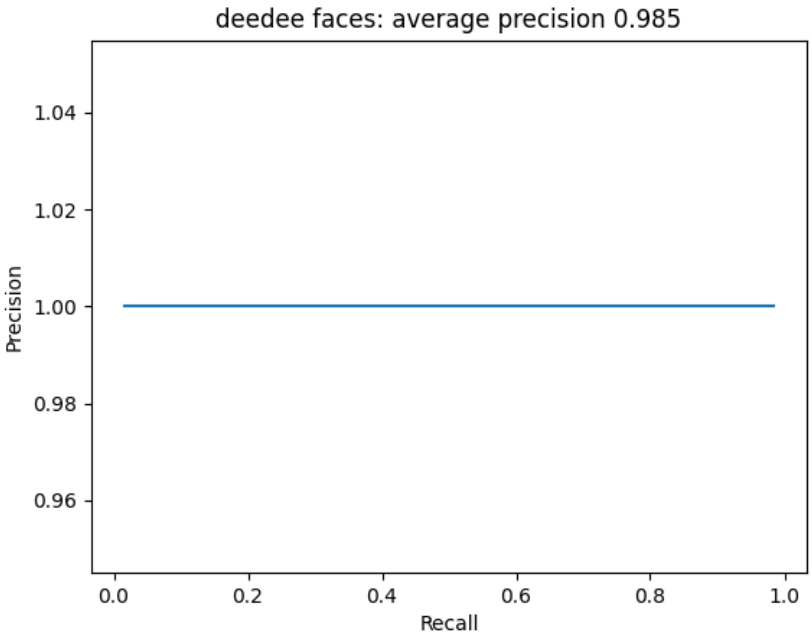
Soluția folosind YOLO pentru DAD



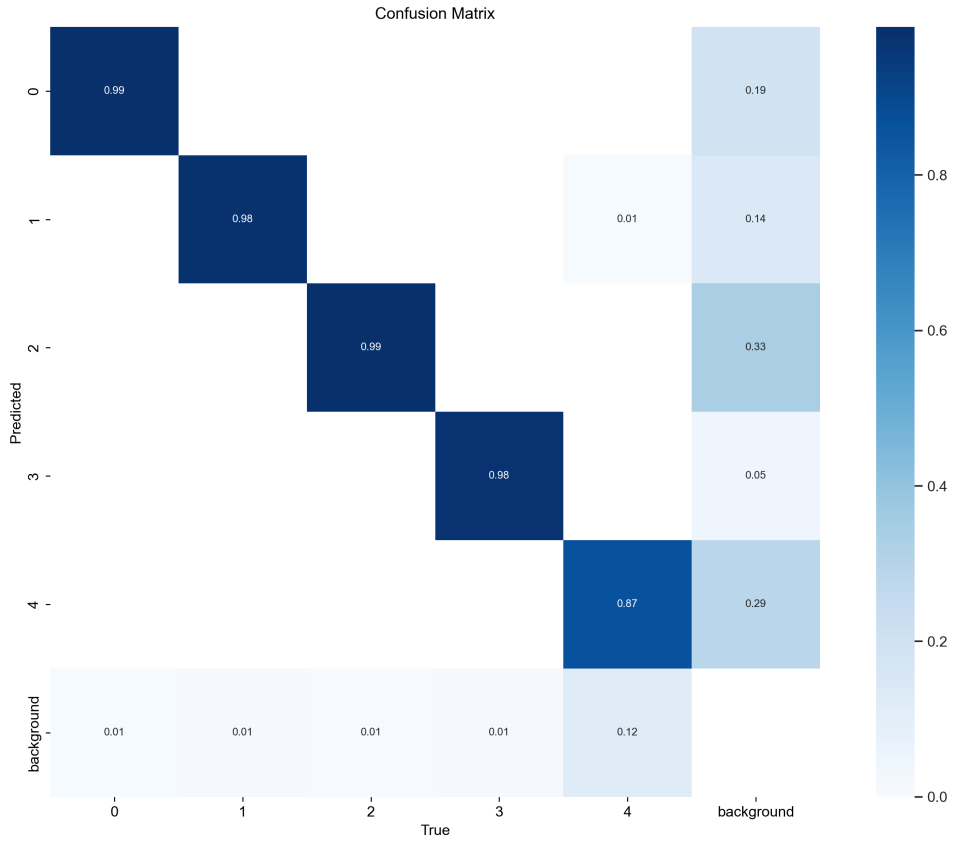
Soluția folosind YOLO pentru MOM



Soluția folosind YOLO pentru DEXTER



Soluția folosind YOLO pentru DEEDEE



Matricea de confuzie pentru modelul YOLO

Putem observa că modelul YOLO se descurcă excelent pe setul nostru de date.

5 Concluzii

Abordarea cu descriptori extrași din AlexNet și clasificatoare neuronale prezintă o metodă clasică dar fiabilă, reușind să ofere o rată de detecție și recunoaștere facială decentă. Deși necesită mai multă putere de calcul (din cauza ferestrelor glisante și a generării descriptorilor pentru fiecare patch), soluția este totuși performantă și își atinge obiectivul de a identifica personajele pe baza caracteristicilor spațiale și vizuale extrase de AlexNet.