Descrierea soluției pentru detectarea și clasificarea personajelor din serialul "Familia Flintstone"

Udrea Iulia-Maria

Grupa 463

1 Introducere

Această documentație prezintă detaliile implementării unor soluții de Computer Vision pentru identificarea și clasificarea personajelor din desenul animat "Familia Flintstone". Pentru a atinge aceste scopuri, proiectul se bazează pe utilizarea a două modele ce folosesc rețele convoluționale: AlexNet, pentru cerințele obligatorii, și Faster R-CNN, pentru cerința bonus.

2 Date și prelucrare

2.1 Extragerea datelor de antrenare

În rezolvarea primei cerințe, am folosit un model AlexNet care diferențiază între fețe și non-fețe. Am extras imagini pozitive folosind bounding-box-urile date pentru fețe și, pentru exemplele negative, am selectat aleatoriu câte trei chenare de 36x36 pixeli din fiecare imagine de antrenare, evitând suprapunerea cu fețele. Datele de antrenare pot fi accesate la adresa: Date Train Drive.

Pentru a doua cerință, am antrenat câte un model AlexNet distinct pentru fiecare personaj principal - Barney, Betty, Fred, Wilma -, având ca obiectiv recunoașterea fiecăruia în imaginile inițial detectate ca fețe. Am folosit ca date de antrenare toate exemplele pozitive din primul task, tratând imaginile cu personajul țintă ca exemple pozitive și restul ca negative.

(a) (b) (c) (d) (e) (f)

Figura 1: Exemple pozitive (a-c) și negative (d-f) pentru task-ul 1

Deoarecea antrenarea modelelor fără date augmentate a avut rezultate mai bune în etapa de testare, modelele finale au fost dezvoltate fără a recurge la augmentarea datelor.

2.2 Prelucrarea datelor

Cu ajutorul datelor de antrenare extrase, se creează o listă de tupluri alcătuite din fiecare imagine și eticheta acesteia. Etichetele sunt stabilite ca 1 pentru exemplele pozitive și 0 pentru cele negative. Imaginile sunt apoi prelucrate pentru a asigura compatibilitatea modelul AlexNet.

Această prelucrare include mai multe transformări:

- **Redimensionarea imaginilor**: Deși dimensiunea originală a imaginilor a fost de 36x36 pixeli, AlexNet nu a funcționat pe dimensiuni atât de mici, astfel încât imaginile au fost transformate la dimensiunea de 64x64 pixeli.
- **Transformarea în tensor și permutarea canalelor**: Imaginile încărcate cu OpenCV sunt în format BGR, în timp ce modelele antrenate PyTorch așteaptă date în format RGB, fiind necesară permutarea canalelor.
- **Normalizarea Imaginilor**: Valorile folosite în normalizare reprezintă media și deviația standard pentru fiecare canal de culoare (RGB) în ImageNet.

Un `data_loader` este utilizat pentru a împărți aceste date transformate în loturi (batch-uri), optimizând procesul de antrenare.

```
def resize_image(image, size=64):
    resized_image = cv.resize(image, (size, size))
    return resized_image

transformed_data = []
for image, label in train_dataset:
    transformed_image = resize_image(image)
    transformed_image = torch.tensor(transformed_image).permute(2, 0, 1).float()
    transformed_image = transforms.Normalize(mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225])(transformed_image)
    transformed_data.append((transformed_image, label))

data_loader = torch.utils.data.DataLoader(transformed_data, batch_size=100,
                    shuffle=True, num_workers=1)
```

Listing 1: Transformarea datelor

Aceste prelucrări sunt aplicate pentru toate datele de antrenare, folosite în ambele cerințe.

3 Antrenarea modelelor

Antrenarea modelelor AlexNet pentru detectarea și clasificarea fețelor este realizată cu ajutorul bibliotecii PyTorch, acestea fiind încărcate fără ponderi pre-antrenate (`pretrained=False`). Modelul trebuie adaptat pentru a lucra cu două clase (fețe și non-fețe, respectiv un anumit personaj vs. restul), deci este necesară ajustarea ultimului strat al modelului.

Se înlocuiește ultimul strat liniar cu unul nou care are același număr de neuroni de intrare (determinați de numărul de filtre din stratul anterior) și 2 neuroni de ieșire, care corespund celor 2 clase.

```
model_alexnet = torch.hub.load('pytorch/vision:v0.10.0', 'alexnet',
                pretrained=False)
nr_filters = model_alexnet.classifier[4].out_features
model_alexnet.classifier[6] = nn.Linear(nr_filters, 2)
# Similar pentru model_barney, model_betty etc.
```

Listing 2: Importarea modelului și modificarea ultimului strat

Antrenarea se realizează la fel pentru toate modelele, fiecare folosind datele specifice. Se utilizează `CrossEntropyLoss` ca funcție de pierdere (loss function) și `SGD` (Stochastic Gradient Descent) ca optimizator, cu o rată de învățare de 0.001 și un momentum de 0.9. Antrenarea de realizează pentru 10 epoci, procesând datele în loturi prin intermediul unui `data_loader` descris anterior.

La fiecare iterație, ponderile modelului sunt actualizate pentru a minimiza funcția de pierdere. Se folosește metoda `backward` pentru a calcula gradientul funcției de pierdere și metoda `step` a optimizatorului pentru a actualiza ponderile. La final se salvează modelele pentru a fi folosite ulterior.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(face_model.parameters(), lr=0.001, momentum=0.9)

for epoch in range(num_epochs):
    for i, data in enumerate(data_loader):
        images, labels = data
        model_alexnet.train()

        optimizer.zero_grad()
        outputs = model_alexnet(images.float())
        _, predicted = torch.max(outputs, 1)

        labels = labels.type(torch.LongTensor)
        labels = labels.to(device)
        loss = criterion(outputs, labels)

        loss.backward()
        optimizer.step()
```

Listing 3: Antrenarea modelului

4 Aplicarea metodei Sliding Window

Dimensiunea ferestrei glisante este stabilită la 64x64 pixeli, aceeași cu dimensiunea imaginilor folosite în antrenarea modelelor. Deoarece fețele personajelor pot avea diferite dimensiuni, imaginea inițială este redimensionată cu diverși factori.

Pentru fiecare versiune redimensionată a imaginii, se extrag ferestre de 64x64 pixeli, cu un pas de 15 pixeli pe ambele direcții, pe verticală și orizontală. Fiecare fereastră este apoi prelucrată pentru a se potrivi cu formatul necesar al modelului de clasificare, ce include normalizarea ferestrei și transformarea ei într-un tensor.

```
for factor in [2, 1.6, 1.2, 1, 0.8, 0.6, 0.4, 0.2]:
        new_img = cv.resize(image.copy(),
        (round(factor*image.shape[1]), round(factor*image.shape[0])))

        for y in range(0, new_img.shape[0] - dim_window, 15):
            for x in range(0, new_img.shape[1] - dim_window, 15):
                img_window = new_img[y:y + dim_window, x:x + dim_window]

                tensor_image = torch.from_numpy(img_window).float()
                tensor_image = tensor_image.permute(2, 0, 1)
                tensor_image = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                std=[0.229, 0.224, 0.225])(tensor_image)
                tensor_image = tensor_image.unsqueeze(0)
```

Listing 4: Extragerea ferestrei și aplicarea transformărilor

Clasificarea se face utilizând primul model, care diferențiază între 'fețe' și 'non-fețe'. Rezultatul clasificării reprezintă două scoruri asociate fiecărei clase, transformate apoi în probabilități folosind funcția `softmax`. Dacă probabilitatea ca fereastra să conțină o față depășește pragul de 0.99999, atunci considerăm că fereastra conține într-adevăr o față. Coordonatele feței detectate sunt calculate în raport cu dimensiunea originală a imaginii, prin împărțirea coordonatelor la factorul de redimensionare.

```
                result = model(tensor_image)
                probabilities = F.softmax(result, dim=1)
                score = probabilities[0, 1].item()

                # Daca scorul depaseste pragul ales, fereastra e considerata fata
                if score > 0.99999:
                    # Determinarea coordonatelor imaginii initiale
                    x_min = int(x // factor)
                    y_min = int(y // factor)
                    x_max = int((x + dim_window) // factor)
                    y_max = int((y + dim_window) // factor)
                    image_detections.append([x_min, y_min, x_max, y_max])
                    image_scores.append(score)
```

Listing 5: Determinarea scorului și a coordonatelor

Pe o fereastră clasificată drept față, aplicăm cei patru clasificatori corespunzători fiecărui personaj - Barney, Betty, Fred, Wilma. Similar cu detectarea feței, se folosește funcția `softmax` pentru a determina probabilitatea fiecărei clase. Dacă nici una dintre clase nu are o probabilitate mai mare de 0.5, clasificăm fereastra ca 'unknown'. Altfel, alegem clasa cu probabilitatea cea mai mare.

```
                    result_barney = model_barney(tensor_image)
                    probabilities_barney = F.softmax(result_barney, dim=1)
                    prob_barney = probabilities_barney[0, 1].item()
                    # Analog pentru toti clasficatorii
```

Listing 6: Determinarea probabilităților pentru fiecare clasă

```
                    # Determinarea clasei in functie de probabilitati
                    if all(prob < 0.5 for prob in
                        [prob_barney, prob_betty, prob_fred, prob_wilma]):
                        predicted_class_name = "unknown"
                    else:
                        max_prob, max_index = max((prob, idx) for idx, prob in
                                enumerate([prob_barney, prob_betty,
                                prob_fred, prob_wilma]))
                        predicted_class_name = class_names[max_index]

                    image_names.append(predicted_class_name)
```

Listing 7: Alegerea clasei în funcție de probabilități

În final, toate detecțiile, scorurile și numele claselor sunt adăugate într-o listă. Pentru a elimina detecțiile redundante sau suprapuse, aplicăm funcția `non_maximal_suppression`, implementată în cadrul laboratorului. Funcția `sliding_window` este apelată pe toate imaginile de testare iar rezultatele sunt salvate în fișiere .npy conform cerinței.

5 Rezolvarea cerinței bonus

Pentru rezolvarea cerințelor la task-ul bonus am ales să folosesc un model Faster-RCNN, care identifică fețele și le clasifică. Implementarea se bazează pe codul de la adresa următoare: Fine-tuning Faster-RCNN using pytorch.

Codul folosește o clasă care extinde `torch.utils.data.Dataset`, pe care am adaptat-o cerințelor proiectului, modificând citirea adnotărilor și clasele (etichetele).

```
self.classes = ['_', 'barney', 'betty', 'fred', 'wilma', 'unknown']
```

Pentru transformarea datelor s-a folosit funcția `ToTensorV2` care convertește o imagine și adnotările sale în tensori PyTorch. Nici în cazul acesta nu am folosit augmentări. Funcția de transformare este aplicată pe datele de antrenare prin intermediul clasei `FlintstoneImagesDataset`, după care sunt împărțite în date de antrenare și de test, folosind `data_loader`.

Se încarcă un model Faster R-CNN preantrenat pe setul de date COCO, adaptat pentru a se potrivi la numărul de clase.

```
def get_object_detection_model(num_classes):
    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)

    # Inlocuirea numarului de clase
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    return model
```

Listing 8: Încărcarea modelului Faster-RCNN

Pentru antrenare, codul utilizează optimizatorul SGD cu o anumită rată de învățare, momentum și weight decay, dar și un `Learning Rate Scheduler` pentru a ajusta rata de învățare pe parcursul antrenării.

```
num_classes = 6
model = get_object_detection_model(num_classes)

params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005,
                            momentum=0.9, weight_decay=0.0005)

lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
```

Listing 9: Alegerea optimizatorului

În fiecare dintre cele 5 epoci, funcția `train_one_epoch` antrenează modelul pe setul de date, iar funcția `evaluate` determină performanța pe setul de test.

```
num_epochs = 5
for epoch in range(num_epochs):
    train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=10)
    lr_scheduler.step()
    evaluate(model, data_loader_test, device=device)
```

Listing 10: Antrenarea modelului Faster-RCNN

La aplicarea modelului pe datele de test, acesta este trecut în modul de evaluare. Se prelucrează datele la fel ca cele de antrenare, iar predicțiile rezultate sunt trecute printr-o funcție de suprimare a non-maximelor.

```
def apply_nms(orig_prediction, iou_thresh=0.3):

    keep = torchvision.ops.nms(orig_prediction['boxes'],
        orig_prediction['scores'], iou_thresh)

    final_prediction = orig_prediction
    final_prediction['boxes'] = final_prediction['boxes'][keep]
    final_prediction['scores'] = final_prediction['scores'][keep]
    final_prediction['labels'] = final_prediction['labels'][keep]

    return final_prediction
```

Listing 11: Aplicarea funcției de suprimare a non-maximelor

Etichetele sunt returnate ca id-uri din lista de clase definită în Dataset, deci trebuie convertite în denumirea acestora înainte de a salva detecțiile în fișierele .npy.

6 Concluzie

Asadar, prin utilizarea acestor clasificatori bazati pe rețele convoluționale, putem aborda cu succes problemele de recunoaștere și clasificare a fețelor din imagini, inclusiv în identificarea personajelor din serialul "Familia Flintstone".