Dokumentation: Gesichtserkennung mit Faster R-CNN und MobileNet v3

Angewandte Modellierung – Projektarbeit

July 7, 2025

Contents

1 Übersicht

Dieses Python-Skript implementiert eine vollständige Pipeline zur **Gesichtserkennung** mithilfe eines **Faster R-CNN** Modells mit MobileNet v3 Backbone. Es wird auf dem **WIDER FACE** Datensatz trainiert und kann auf eigene Bilder angewendet werden.

2 Voraussetzungen

2.1 Bibliotheken

- `torch`
- `torchvision`
- `Pillow`
- `scipy`
- `numpy`

2.2 Hardware

GPU-Unterstützung wird empfohlen, aber das Skript läuft auch auf CPU (langsamer).

3 Projektstruktur

```
project_root/

  data/
      widerface/
          WIDER_train/WIDER_train/images/
          wider_face_annotations/wider_face_split/wider_face_train.mat

  checkpoints/
  inference_results/
  own_images/
  image_detect.py
```

4 Wichtige Komponenten des Skripts

4.1 ObjectDetectionDataset

Eine benutzerdefinierte PyTorch Dataset-Klasse zur Vorbereitung der Trainingsdaten:

```
class ObjectDetectionDataset(torch.utils.data.Dataset):
    def __init__(self, annotations, transforms=None):
        ...
```

4.2 Transformationsfunktion

Gibt transformationsbasierte Datenaugmentierung für Training zurück:

```
def get_transforms(train):
    transforms_list = [T.ToTensor()]
    if train:
        transforms_list.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms_list)
```

4.3 Annotation-Parser

Lädt Daten aus .mat-Datei und extrahiert Bounding-Boxes und Labels:

```
def load_wider_annotations(mat_path, images_root):
    data = scipy.io.loadmat(mat_path)
    ...
    return records
```

5 Trainingspipeline

1. Daten laden mit `ObjectDetectionDataset`
2. Faster R-CNN Modell mit MobileNet v3 Initialisierung
3. Modifikation des Klassifizierers auf 2 Klassen (Hintergrund, Gesicht)
4. Optimizer: SGD mit Lernraten-Scheduler
5. 20 Epochen Training mit Verlustberechnung und Scheduler-Step

5.1 Modellinitialisierung

```
model = fasterrcnn_mobilenet_v3_large_fpn(weights=None)
num_classes = 2
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
```

5.2 Training Loop

```
for epoch in range(num_epochs):
    model.train()
    ...
    print(f"Epoch [{epoch+1}/{num_epochs}] Loss: {epoch_loss/len(
        train_loader):.4f}")
```

5.3 Checkpoint-Speicherung

```
torch.save(model.state_dict(), checkpoint_path)
```

6 Inference / Vorhersage

Mit der Funktion `run_inference` können beliebige Bilder durch das trainierte Modell analysiert und mit Bounding Boxes versehen werden.

6.1 Beispiel-Aufruf

```
custom_images = ["path/to/image1.jpg", "path/to/image2.jpg"]
run_inference(model, custom_images, inference_dir, device)
```

6.2 Funktionsweise

- Öffnet Bild mit PIL
- Wandelt es in Tensor um
- Führt Modellinferenz aus
- Zeichnet erkannte Gesichter mit Bounding Boxes
- Speichert Bild mit Ergebnis in `inference_results`

7 Modellspeicherung und Wiederverwendung

Das Modell wird automatisch gespeichert und beim nächsten Start wieder geladen:

```
checkpoint_path = checkpoints_dir / "fasterrcnn_mobilenet_v3_finetuned.pth"
if checkpoint_path.exists():
    model.load_state_dict(torch.load(checkpoint_path, map_location=device))
```

8 Anpassungsmöglichkeiten

- **Eigene Bilder:** Bildpfade in `custom_images` anpassen
- **Modellarchitektur:** Alternative wie `fasterrcnn_resnet50_fpn`
- **Weitere Objektklassen:** Labels und `num_classes` erweitern

9 Beispielausgabe

```
Loaded 12880 training images
Epoch [1/20] Loss: 1.3482
...
Saved inference result to inference_results/people.jpg
```

10 Fazit

Dieses Skript bietet eine robuste, GPU-optimierte Lösung für das Training und die Anwendung eines Gesichtserkennungsmodells. Durch die modulare Struktur ist es leicht erweiterbar für allgemeine Objekterkennung mit PyTorch.