Code Documentation

Overview

This software implements a two-model approach for person detection in images, focusing specifically on thermal imagery. The system first uses a primary model (Model 1) to detect people, and based on confidence thresholds, it either accepts the detection, runs a secondary verification model (Model 2), or rejects the detection entirely. The system incorporates advanced techniques including data augmentation and inpainting for improved detection performance.

Model Architecture

Model 1: Person-Only Detection

- **Purpose**: Primary detection of people in thermal images
- **Classes**: 1 class ('person')
- **Training Data**: Located at D:/project_work/model1/dataset/train
- **Validation Data**: Located at D:/project_work/model1/dataset/valid
- **Test Data**: Located at D:/project_work/model1/dataset/test

Model 2: Body Part Detection

- **Purpose**: Secondary detection and verification of people by identifying body parts
- **Classes**: 5 classes ('arm', 'head', 'leg', 'person', 'torso')
- **Training Data**: Located at D:/project_work/model_3/dataset/train
- **Validation Data**: Located at D:/project_work/model_3/dataset/valid
- **Test Data**: Located at D:/project_work/model_3/dataset/test

Software Requirements

Core Dependencies

- Python 3.7+
- PyTorch 1.7.0+
- TorchVision 0.8.1+
- OpenCV (cv2) 4.5.0+
- NumPy 1.19.0+
- YOLOv5 (will be installed via torch.hub)

Pre-trained Models

- Model 1: YOLOv5 model trained for person detection
  - Path: D:/project_work/model_1/yolov5/runs/train/model1_train2/weights/best.pt
- Model 2: YOLOv5 model trained for body part detection
  - Path: D:/project_work/model_3/yolov5/runs/train/exp3/weights/best.pt

Data Preparation & Augmentation

Data Augmentation Techniques

Both models utilized extensive data augmentation as specified in their YAML configuration files:

- **Color Transformations**:
  - HSV-Hue: ±0.015 fraction
  - HSV-Saturation: ±0.7 fraction
  - HSV-Value: ±0.4 fraction
- **Geometric Transformations**:
  - Translation: ±0.1 fraction
  - Scale: ±0.5 gain
  - Left-right flip: 0.5 probability
  - Mosaic augmentation: 1.0 probability (always applied)

Inpainting

The system incorporates inpainting techniques to enhance detection performance:

- Inpainted images were added to the training data
- The test example (inpainted_result1.jpg) is an inpainted thermal image
- Inpainting helps reduce noise and enhance person features in thermal imagery

Model Training Configurations

Common Training Parameters

Both models were trained with similar hyperparameters:

- **Image Size**: 640×640 pixels
- **Batch Size**: 16
- **Training Epochs**: 150
- **Patience**: 100 epochs

- **Optimizer**: SGD
    - Initial learning rate: 0.01
    - Final learning rate factor: 0.01
    - Momentum: 0.937
    - Weight decay: 0.0005
- **Warmup**:
    - Epochs: 3.0
    - Initial momentum: 0.8
    - Initial bias learning rate: 0.1

Model-Specific Parameters

- **Frozen Layers**: First layer [0] frozen during training
- **Loss Parameters**:
    - Box loss gain: 0.05
    - Classification loss gain: 0.5
    - Object loss gain: 1.0
    - IoU training threshold: 0.20

Ensemble Detection System

Logic Flow

The system operates based on the following confidence thresholds:

1. **Model 1 Detection**:
    - If confidence > 90%: Accept detection, display result, and exit
    - If confidence between 85-90%: Output "person not detected properly" and run Model 3
    - If confidence < 85%: Run Model 3 without notification
2. **Model 2 Detection** (when needed):
    - If confidence ≥ 85%: Accept detection and display result
    - If confidence < 85% or no detection: Output "no living person detected"

Detection Parameters

- **Confidence Threshold**: 0.25 (during model training)
- **IoU Threshold**: 0.4 (for Non-Maximum Suppression)
- **Maximum Detections**: 300 per image

Setup Instructions

Install Python Dependencies:

```
pip install torch torchvision opencv-python numpy
```

1. pip install torch torchvision opencv-python numpy
2. **Model Paths**: Ensure both models are available at the specified paths:

   - Model 1: D:/project_work/model_1/yolov5/runs/train/model1_train2/weights/best.pt
   - Model 2: D:/project_work/model_3/yolov5/runs/train/exp3/weights/best.pt
3. Update the paths in the code if your models are stored in different locations.

4. **Test Image**: Place your test image at the specified path or update the image path in the code:

   - Default path: D:/project_work/model_3/inpainting/inpainted_result1.jpg

Execution Instructions

Running the Program:

```
python person_detection.py
```

1. python person_detection.py
2. **Output Interpretation**:

   - The program will display console messages indicating detection confidence
   - When a person is successfully detected, a window will appear showing the image with bounding boxes
   - Results will be saved as image files in the current directory
3. **Result Files**:

   - high_confidence_result.jpg: Results from Model 1 with high confidence
   - model2_verification_result.jpg: Results from Model 2 when used for verification
   - model2_result.jpg: Results from Model 2 when used as primary detector

Code Structure

Main Components

1. **Model Loading Functions**:

   - run_model1(): Loads and runs the primary person detection model
   - run_model2(): Loads and runs the secondary verification model (Model 2)
2. **Visualization Function**:

   - draw_box(): Handles drawing bounding boxes with labels on detected persons

3. **Main Execution Function**:

   - main(): Orchestrates the detection workflow based on confidence thresholds

Complete Code

```
import torch
import numpy as np
import cv2
from torchvision.ops import nms

def run_model1(img_path):
    # Load model1 which detects people
    model1 = torch.hub.load('ultralytics/yolov5', 'custom',
path='D:/project_work/model_1/yolov5/runs/train/model1_train2/weights/best.pt')
    model1.eval()

    # Run inference
    results1 = model1(img_path)

    # Get predictions from model 1
    boxes1 = results1.xyxy[0].cpu().numpy()  # Using xyxy format

    # Filter for only person class if needed
    # Assuming "person" is one of the classes - adjust class_id if different
    person_boxes = []
    person_scores = []

    for box in boxes1:
        class_id = int(box[5])
        if results1.names[class_id] == 'person':  # Filter for person class
            person_boxes.append(box[:4])  # x1, y1, x2, y2
            person_scores.append(box[4])  # Confidence score

    # If no persons detected
    if not person_scores:
        return None, None, None

    # Find the highest confidence score
    max_score = max(person_scores) if person_scores else 0
    max_index = person_scores.index(max_score) if person_scores else -1

    if max_index != -1:
        return max_score, person_boxes[max_index], results1.names
    else:
        return None, None, None

def run_model2(img_path):
    # Load model2 which is the backup model (Model 3)
    model2 = torch.hub.load('ultralytics/yolov5', 'custom',
path='D:/project_work/model_3/yolov5/runs/train/exp3/weights/best.pt')
    model2.eval()

    # Run inference
    results2 = model2(img_path)

    # Get predictions from model 2
    boxes2 = results2.xyxy[0].cpu().numpy()

    # Filter for only person class if needed
    person_boxes = []
    person_scores = []

    for box in boxes2:
        class_id = int(box[5])
        if results2.names[class_id] == 'person':  # Filter for person class
            person_boxes.append(box[:4])  # x1, y1, x2, y2
            person_scores.append(box[4])  # Confidence score

    # If no persons detected
    if not person_scores:
        return None, None, None

    # Find the highest confidence score
    max_score = max(person_scores) if person_scores else 0
    max_index = person_scores.index(max_score) if person_scores else -1

    if max_index != -1:
        return max_score, person_boxes[max_index], results2.names
    else:
        return None, None, None

def draw_box(image, box, label, score):
    image_height, image_width = image.shape[:2]
    x1, y1, x2, y2 = int(box[0]), int(box[1]), int(box[2]), int(box[3])

    # Ensure coordinates are within image bounds
    x1 = max(0, min(x1, image_width - 1))
    y1 = max(0, min(y1, image_height - 1))
    x2 = max(0, min(x2, image_width - 1))
    y2 = max(0, min(y2, image_height - 1))

    cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)  # Draw green rectangle

    # Add label and confidence score
    label_text = f"{label}: {score:.2f}"
    cv2.putText(image, label_text, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    return image

def main():
    # Example image path
    img_path = 'D:/project_work/model_3/inpainting/inpainted_result1.jpg'

    # Load the image
    image = cv2.imread(img_path)
    if image is None:
        print(f"Error: Could not load image at {img_path}")
        return

    # First, run model 1
    print("Running model 1 (person detection)...")
    conf_score1, best_box1, names1 = run_model1(img_path)

    if conf_score1 is None:
        print("No person detected by model 1.")
        print("No living person detected.")
        return

    # Check confidence thresholds for model 1
    if conf_score1 >= 0.90:  # Above 90%
        print(f"Person detected with high confidence: {conf_score1:.2f}")
        # Draw the bounding box
        result_image = draw_box(image.copy(), best_box1, "person", conf_score1)
        cv2.imshow('Person Detection Result (Model 1)', result_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
        cv2.imwrite('high_confidence_result.jpg', result_image)

    elif 0.85 <= conf_score1 < 0.90:  # Between 85-90%
        print(f"Person not detected properly in model 1. Confidence: {conf_score1:.2f}")
        print("Running model 2 to verify...")

        # Run model 2 for verification
        conf_score2, best_box2, names2 = run_model2(img_path)

        if conf_score2 is not None:
            print(f"Person detected by model 2 with confidence: {conf_score2:.2f}")
            # Draw the bounding box from model 2
            result_image = draw_box(image.copy(), best_box2, "person", conf_score2)
            cv2.imshow('Person Detection Result (Model 2)', result_image)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
            cv2.imwrite('model2_verification_result.jpg', result_image)
        else:
            print("No person detected by model 2.")
            print("No living person detected.")

    else:  # Below 85%
        print(f"Low confidence person detection: {conf_score1:.2f}")
        print("Running model 2 for verification...")

        # Run model 2
        conf_score2, best_box2, names2 = run_model2(img_path)

        if conf_score2 is not None and conf_score2 >= 0.85:
            print(f"Person detected by model 2 with confidence: {conf_score2:.2f}")
            # Draw the bounding box from model 2
            result_image = draw_box(image.copy(), best_box2, "person", conf_score2)
            cv2.imshow('Person Detection Result (Model 2)', result_image)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
            cv2.imwrite('model2_result.jpg', result_image)
        else:
            # Either no person detected or confidence below 0.85
            print("No living person detected.")

if __name__ == "__main__":
    main()
```

Model YAML Configurations

Model 1 Configuration (data.yaml)

```
# YOLOv5 Thermal Body Part Detection Configuration

# Train/val/test sets
train: D:/project_work/model1/dataset/train
val: D:/project_work/model1/dataset/valid
test: D:/project_work/model1/dataset/test

# Classes
nc: 1  # number of classes
names: ['person']  # class names

# Hyperparameters optimized for thermal body part detection
hyp:
  # Augmentation parameters
  hsv_h: 0.015  # HSV-Hue augmentation (fraction)
  hsv_s: 0.7    # HSV-Saturation augmentation (fraction)
  hsv_v: 0.4    # HSV-Value augmentation (fraction)
  degrees: 0.0  # image rotation (+/- deg)
  translate: 0.1  # image translation (+/- fraction)
  scale: 0.5    # image scale (+/- gain)
  shear: 0.0    # image shear (+/- deg)
  perspective: 0.0  # image perspective (+/- fraction)
  flipud: 0.0   # image flip up-down (probability)
  fliplr: 0.5   # image flip left-right (probability)
  mosaic: 1.0   # image mosaic (probability)
  mixup: 0.0    # image mixup (probability)

  # Model parameters
  box: 0.05     # box loss gain
  cls: 0.5      # cls loss gain
  cls_pw: 1.0   # cls BCELoss positive_weight
  obj: 1.0      # obj loss gain
  obj_pw: 1.0   # obj BCELoss positive_weight
  iou_t: 0.20   # IoU training threshold
  anchor_t: 4.0 # anchor-multiple threshold
  fl_gamma: 0.0 # focal loss gamma

  # Anchors
  anchors: 3    # number of anchors per scale

# Detection parameters
conf_thres: 0.25  # confidence threshold
iou_thres: 0.4    # NMS IoU threshold
max_det: 300      # maximum detections per image
```

```
# Image size and batch parameters
img_size: 640     # image size
batch_size: 16    # batch size

# Training parameters
epochs: 150       # number of epochs
patience: 100     # epochs to wait for no observable improvement
freeze: [0]       # freeze first layer during training

# Optimizer parameters
optimizer: SGD    # optimizer: ['SGD', 'Adam', 'AdamW']
lr0: 0.01         # initial learning rate
lrf: 0.01         # final learning rate (lr0 * lrf)
momentum: 0.937   # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay
warmup_epochs: 3.0    # warmup epochs
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1   # warmup initial bias lr
```

Model 2 Configuration (data.yaml)

```
# YOLOv5 Thermal Body Part Detection Configuration
# Train/val/test sets
train: D:/project_work/model_3/dataset/train
val: D:/project_work/model_3/dataset/valid
test: D:/project_work/model_3/dataset/test
# Classes
nc: 5  # number of classes
names: ['arm', 'head', 'leg', 'person', 'torso']  # class names
# Hyperparameters optimized for thermal body part detection
hyp:
  # Augmentation parameters
  hsv_h: 0.015  # HSV-Hue augmentation (fraction)
  hsv_s: 0.7    # HSV-Saturation augmentation (fraction)
  hsv_v: 0.4    # HSV-Value augmentation (fraction)
  degrees: 0.0  # image rotation (+/- deg)
  translate: 0.1  # image translation (+/- fraction)
  scale: 0.5    # image scale (+/- gain)
  shear: 0.0    # image shear (+/- deg)
  perspective: 0.0  # image perspective (+/- fraction)
  flipud: 0.0   # image flip up-down (probability)
  fliplr: 0.5   # image flip left-right (probability)
  mosaic: 1.0   # image mosaic (probability)
  mixup: 0.0    # image mixup (probability)
  # Model parameters
  box: 0.05     # box loss gain
  cls: 0.5      # cls loss gain
  cls_pw: 1.0   # cls BCELoss positive_weight
  obj: 1.0      # obj loss gain
  obj_pw: 1.0   # obj BCELoss positive_weight
  iou_t: 0.20   # IoU training threshold
  anchor_t: 4.0 # anchor-multiple threshold
  fl_gamma: 0.0 # focal loss gamma
  # Anchors
  anchors: 3    # number of anchors per scale
# Detection parameters
conf_thres: 0.25  # confidence threshold
iou_thres: 0.4    # NMS IoU threshold
max_det: 300      # maximum detections per image
# Image size and batch parameters
img_size: 640     # image size
batch_size: 16    # batch size
# Training parameters
epochs: 150       # number of epochs
patience: 100     # epochs to wait for no observable improvement
freeze: [0]       # freeze first layer during training
# Optimizer parameters
optimizer: SGD    # optimizer: ['SGD', 'Adam', 'AdamW']
lr0: 0.01         # initial learning rate
lrf: 0.01         # final learning rate (lr0 * lrf)
momentum: 0.937   # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay
warmup_epochs: 3.0    # warmup epochs
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1   # warmup initial bias lr
```

Customization Options

Adjusting Confidence Thresholds

Modify these values in the main() function to adjust sensitivity:

- High confidence threshold (currently 0.90)
- Medium confidence threshold (currently 0.85)

Using Different Models

To use different YOLOv5 models:

1. Update the model paths in the `run_model1()` and `run_model2()` functions
2. Ensure the models are trained to detect the appropriate classes

Processing Multiple Images

To process multiple images:

1. Create a list of image paths
2. Loop through the list and call `main()` for each image
3. Modify the output file naming scheme to avoid overwriting

Troubleshooting

Common Issues and Solutions

1. **Model Loading Error**:
   - Ensure the model paths are correct
   - Verify you have internet access for the first run (YOLOv5 will download dependencies)
2. **No Detection**:
   - Check if the image contains clearly visible persons
   - Adjust confidence thresholds if detections are being filtered out
3. **CUDA/GPU Issues**:
   - Add `device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')`
   - Pass `device` parameter to model loading functions
4. **Memory Issues**:
   - Reduce image size before processing
   - Use model.half() for reduced memory usage with slight accuracy loss
5. **Inpainting Issues**:
   - Ensure inpainted images maintain the thermal characteristics
   - Check that inpainting doesn't remove critical features for detection