GNN + Transformer for Multimodal Keypoint-Based Score Generation

This project implements a Graph Neural Network (GNN) + Transformer model for generating scores using multimodal keypoint data from body, hand, and object tracking.

Overview

The system processes multimodal data from different sources:

1. OpenPose body keypoints (shoulders, elbows, wrists, neck, mid hip)
2. Google MediaPipe hand keypoints (fingertips and wrist)
3. Object location data from TridentNet

The model combines spatial processing (via GNN) and temporal processing (via Transformer) to generate accurate scores for human activities.

Project Structure

```
├── multi_pickle_processor.py   # Processes data from multiple pickle directories
├── data_loader.py              # Loads and processes data for the model
├── gnn_transformer_updated.py  # Main GNN + Transformer model implementation
├── main_script.py              # Training, testing and evaluation functions
├── run_pipeline.py             # End-to-end pipeline runner
├── README.md                   # This file
```

Data Structure

The system expects data in the following directories:

- `D:/pickle_dir`: Main body keypoints from OpenPose
- `D:/pickle_files`: Alternative OpenPose format
- `D:/pickle_files_hand`: Hand keypoints from MediaPipe
- `D:/pickle_files_object`: Object locations from TridentNet
- `D:/pickle_dir/therapist_labels.csv`: Therapist labels for scoring

Requirements

- Python 3.8+

- PyTorch 1.10+
- PyTorch Geometric (for GNN operations)
- NumPy, Pandas, Matplotlib, tqdm, scikit-learn
- OpenCV (for visualization)

Installation

```bash
# Create and activate a conda environment
conda create -n gnn_transformer python=3.8
conda activate gnn_transformer

# Install PyTorch (adjust version for CUDA compatibility)
conda install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch

# Install PyTorch Geometric
conda install pyg -c pyg

# Install other dependencies
pip install numpy pandas matplotlib tqdm scikit-learn opencv-python
```

Usage

Running the Full Pipeline

The easiest way to run the pipeline is to use the `run_pipeline.py` script:

```python
python run_pipeline.py [mode]
```

Available modes:

- `process_only`: Only process pickle files and create segment database
- `train_only`: Only train model using existing segment database
- `process_and_train`: Process pickle files and train model (default)
- `cross_validate`: Process pickle files and run cross-validation

Processing Data Only

If you only want to process the data and create the segment database:

python

```python
from multi_pickle_processor import MultiPickleProcessor

processor = MultiPickleProcessor(
    pickle_dirs={
        'body': 'D:/pickle_dir',
        'openpose': 'D:/pickle_files',
        'hand': 'D:/pickle_files_hand',
        'object': 'D:/pickle_files_object'
    },
    output_dir='D:/combined_segments',
    num_files_per_dir=10
)

processor.process(
    therapist_labels_path='D:/pickle_dir/therapist_labels.csv',
    output_filename='segment_database.pkl'
)
```

Training the Model

To train the model using a pre-processed segment database:

python

```python
from main_script import train_model

train_model(
    segment_db_path='D:/combined_segments/segment_database.pkl',
    output_dir='./output/gnn_transformer',
    view_type='top',
    epochs=30,
    batch_size=8,
    balance_classes=True
)
```

Cross-Validation

To evaluate the model using cross-validation:

```python
from main_script import cross_validate

cross_validate(
    segment_db_path='D:/combined_segments/segment_database.pkl',
    view_type='top',
    output_dir='./output/gnn_transformer_cv',
    num_folds=5
)
```

Visualization

The model includes visualization capabilities for keypoints and attention weights:

```python
from main_script import visualize_keypoints, visualize_attention

# Load model and data first, then:
visualize_keypoints(test_loader, output_dir='./output/visualizations')
visualize_attention(model, test_loader, device, output_dir='./output/visualizations')
```

Model Architecture

The model architecture consists of:

1. **Graph Construction**: Creates a graph from keypoints with nodes representing body joints, hand keypoints, and object locations.
2. **Graph Neural Network (GNN)**: Processes spatial relationships between keypoints within each frame.
   - Uses GCN (Graph Convolutional Network) layers to exchange information between connected keypoints
   - Captures spatial dependencies between body, hands, and objects
3. **Transformer Encoder**: Processes temporal relationships across frames.
   - Applies self-attention to learn which frames are most important for scoring
   - Captures long-range dependencies in the sequence
4. **Classification Head**: Generates the final score prediction.

Configuration Options

The model has many configuration options that can be set in the `run_pipeline.py` script:

- **Data Configuration**:
  - `pickle_dirs`: Paths to pickle directories
  - `output_dir`: Output directory for segment database
  - `num_files_per_dir`: Number of pickle files to process from each directory
- **Model Configuration**:
  - `view_type`: Camera view to use ('top' or 'ipsilateral')
  - `seq_length`: Maximum sequence length
  - `gnn_hidden`: Hidden dimension of GNN layers
  - `gnn_out`: Output dimension of GNN embedding
  - `transformer_heads`: Number of transformer attention heads
  - `transformer_layers`: Number of transformer layers
  - `dropout`: Dropout rate
- **Training Configuration**:
  - `epochs`: Number of training epochs
  - `batch_size`: Batch size
  - `lr`: Learning rate
  - `weight_decay`: Weight decay coefficient
  - `balance_classes`: Whether to balance classes in training
  - `seed`: Random seed

Output Files

The model generates various output files:

- **Model Checkpoints**:
  - `gnn_transformer_best.pt`: Best model checkpoint
  - `gnn_transformer_epoch_X.pt`: Checkpoints at regular intervals
- **Evaluation Results**:
  - `test_results.json`: Test metrics and results
  - `confusion_matrix.png`: Confusion matrix visualization
  - `training_curves.png`: Learning curves
- **Visualizations**:

- Keypoint visualizations
- Attention weight visualizations

Notes on Data Processing

- **Patient-Level Organization**: Data is organized by patient ID and activity ID
- **Segment-Level Processing**: Segments are created based on therapist ratings
- **Camera Views**: Can use either top view (cam3) or ipsilateral view
- **Therapist Ratings**: Ratings are mapped to binary classification labels (0 or 1)

Future Work

- Integration with real-time keypoint extraction systems
- Extension to handle more complex activities and tasks
- Exploration of different GNN architectures (GAT, GraphSAGE, etc.)
- Implementation of attention visualization for better interpretability

License

This project is for research purposes only.

Acknowledgements

This project builds on:

- PyTorch Geometric for GNN operations
- OpenPose for body keypoint extraction
- MediaPipe for hand keypoint extraction
- TridentNet for object detection