Pose 및 STGCN 학습 설정 설명 및 가이드

개요

이 문서는 Recognizer 프로젝트에서 사용하는 포즈 추정 모델(RTMO)과 동작 분류 모델(STGCN++)의 학습 설정 및 최적화 방법을 설명합니다.

1. 포즈 추정 모델 (RTMO) 설정

1.1 RTMO 모델 개요

- **모델명**: Real-Time Multi-Object (RTMO)
- **기반**: MMPose 프레임워크
- **특징**: 실시간 다중 인물 포즈 추정
- **출력**: COCO17 형식 17개 키포인트

1.2 지원 백엔드

PyTorch 백엔드

```yaml
models:
  pose_estimation:
    inference_mode: pth
    pth:
      config_file: /workspace/mmpose/configs/body_2d_keypoint/rtmo/body7/rtmo-l_16xb16-600e_body7-640x640.py
      checkpoint_path: /workspace/mmpose/checkpoints/rtmo-l_16xb16-600e_body7-640x640-b37118ce_20231211.pth
      device: cuda:0
      input_size: [640, 640]
      score_threshold: 0.2
      nms_threshold: 0.65
      keypoint_threshold: 0.3
```

ONNX 백엔드 (권장)

```yaml
models:
  pose_estimation:
    inference_mode: onnx
    onnx:
      model_path: /workspace/mmpose/checkpoints/end2end_l.onnx
      device: cuda:0
      model_input_size: [640, 640]
      score_threshold: 0.3
      nms_threshold: 0.45
      keypoint_threshold: 0.3
      max_detections: 100

      # ONNX Runtime 최적화
      execution_mode: ORT_SEQUENTIAL
      graph_optimization_level: ORT_ENABLE_ALL
      enable_mem_pattern: true
      enable_cpu_mem_arena: true
      gpu_mem_limit_gb: 8
```

TensorRT 백엔드 (최고 성능)

```yaml
models:
  pose_estimation:
    inference_mode: tensorrt
    tensorrt:
      model_path: /workspace/mmpose/checkpoints/rtmo.trt
      device: cuda:0
      model_input_size: [640, 640]
      score_threshold: 0.3
      nms_threshold: 0.45
      fp16_mode: false
```

1.3 모델 변환 과정

PyTorch → ONNX 변환

```bash
# MMPose 모델을 ONNX로 변환
cd /workspace/mmpose
python tools/deployment/pytorch2onnx.py \
    configs/body_2d_keypoint/rtmo/body7/rtmo-l_16xb16-600e_body7-640x640.py \
    checkpoints/rtmo-l_16xb16-600e_body7-640x640-b37118ce_20231211.pth \
    --output-file checkpoints/end2end_l.onnx \
    --input-img demo/demo.jpg \
    --test-img demo/demo.jpg \
    --dynamic-export \
    --verify
```

ONNX → TensorRT 변환

```
# TensorRT 엔진 생성
cd /workspace/mmpose
python tools/deployment/onnx2tensorrt.py \
    checkpoints/end2end_l.onnx \
    --output checkpoints/rtmo.trt \
    --input-img demo/demo.jpg \
    --min-shape 1 3 640 640 \
    --opt-shape 1 3 640 640 \
    --max-shape 1 3 640 640 \
    --fp16
```

1.4 성능 최적화 설정

GPU 메모리 최적화

```
onnx:
  gpu_mem_limit_gb: 8               # GPU 메모리 제한
  arena_extend_strategy: kNextPowerOfTwo
  enable_cpu_mem_arena: true
  enable_mem_pattern: true
```

처리 속도 최적화

```
onnx:
  execution_mode: ORT_SEQUENTIAL    # 또는 ORT_PARALLEL
  graph_optimization_level: ORT_ENABLE_ALL
  tunable_op_enable: true
  tunable_op_tuning_enable: true
  cudnn_conv_algo_search: HEURISTIC
```

2. STGCN++ 동작 분류 모델 설정

2.1 모델 아키텍처

기본 STGCN++ 구조

```
model = dict(
    backbone=dict(
        type='STGCN',
        gcn_adaptive='init',              # 적응적 그래프 학습
        gcn_with_res=True,                # 잔차 연결 사용
        tcn_type='mstcn',                 # Multi-Scale TCN
        graph_cfg=dict(
            layout='coco',                # COCO17 키포인트 레이아웃
            mode='spatial'                # 공간적 그래프 구성
        ),
        num_stages=4,                     # 네트워크 단계 수
        base_channels=64,                 # 기본 채널 수
        inflate_stages=[2],               # 채널 확장 단계
        down_stages=[2],                  # 다운샘플링 단계
    ),
    cls_head=dict(
        type='GCNHead',
        num_classes=2,                    # Fight/NonFight 또는 Normal/Falldown
        in_channels=128,                  # 백본 출력 채널
        dropout=0.3,                      # 드롭아웃 비율
        loss_cls=dict(
            type='CrossEntropyLoss',
            class_weight=[1.2, 1.0]      # 클래스 불균형 조정
        )
    )
)
```

2.2 데이터 파이프라인

훈련 파이프라인

```
train_pipeline = [
    dict(type='PreNormalize2D'),                    # 좌표 정규화
    dict(type='GenSkeFeat', dataset='coco', feats=['b']),  # 본 특징 생성
    dict(type='UniformSampleFrames', clip_len=100), # 균등 프레임 샘플링
    dict(type='PoseDecode'),                        # 포즈 디코딩
    dict(type='FormatGCNInput', num_person=4),      # GCN 입력 형식화
    dict(type='PackActionInputs')                   # 액션 입력 패킹
]
```

검증/테스트 파이프라인

```
val_pipeline = [
    dict(type='PreNormalize2D'),
    dict(type='GenSkeFeat', dataset='coco', feats=['b']),
    dict(type='UniformSampleFrames', clip_len=100, num_clips=1, test_mode=True),
    dict(type='PoseDecode'),
    dict(type='FormatGCNInput', num_person=4),
    dict(type='PackActionInputs')
]
```

2.3 Fight 감지 모델 설정

데이터셋 경로

```
dataset_type = 'PoseDataset'
data_root = '/workspace/recognizer/test_data'
ann_file_train = '/workspace/recognizer/output/RWF-2000/stage3_dataset/.../train.pkl'
ann_file_val = '/workspace/recognizer/output/RWF-2000/stage3_dataset/.../val.pkl'
ann_file_test = '/workspace/recognizer/output/RWF-2000/stage3_dataset/.../test.pkl'
```

클래스 가중치 설정

```
loss_cls=dict(
    type='CrossEntropyLoss',
    class_weight=[1.2, 1.0]  # NonFight:Fight = 45.4:54.6 분포 반영
)
```

2.4 Falldown 감지 모델 설정

데이터셋 경로

```
ann_file_train = '/workspace/recognizer/output/falldown_aihub/stage3_dataset/.../train.pkl'
ann_file_val = '/workspace/recognizer/output/falldown_aihub/stage3_dataset/.../val.pkl'
ann_file_test = '/workspace/recognizer/output/falldown_aihub/stage3_dataset/.../test.pkl'
```

Falldown 특화 설정

```
# Falldown 감지는 더 민감한 설정 필요
model = dict(
    cls_head=dict(
        dropout=0.2,  # 더 낮은 드롭아웃
        loss_cls=dict(
            type='CrossEntropyLoss',
            class_weight=[1.0, 1.5]  # Falldown 클래스에 더 높은 가중치
        )
    )
)
```

3. 학습 설정 및 하이퍼파라미터

3.1 옵티마이저 설정

AdamW 옵티마이저 (권장)

```
optim_wrapper = dict(
    type='OptimWrapper',
    optimizer=dict(
        type='AdamW',
        lr=0.0001,                    # 학습률
        weight_decay=0.001,           # 가중치 감쇠
        betas=(0.9, 0.999),          # Adam 베타 파라미터
        eps=1e-8
    ),
    clip_grad=dict(max_norm=2.0, norm_type=2)  # 그래디언트 클리핑
)
```

SGD 옵티마이저 (대안)

```
optim_wrapper = dict(
    type='OptimWrapper',
    optimizer=dict(
        type='SGD',
        lr=0.01,
        momentum=0.9,
        weight_decay=0.0005
    )
)
```

3.2 학습률 스케줄러

Linear Warmup + MultiStep 스케줄러 (안정적)

```
param_scheduler = [
    dict(
        type='LinearLR',
        start_factor=0.1,             # 시작 학습률 비율
        by_epoch=True,
        begin=0,
        end=3,                        # 3 에폭 동안 warmup
        convert_to_iter_based=True
    ),
    dict(
        type='MultiStepLR',
        by_epoch=True,
        begin=3,
        milestones=[15, 25],          # 15, 25 에폭에서 학습률 감소
        gamma=0.5                     # 감소 비율
    )
]
```

Cosine Annealing 스케줄러 (실험적)

```
param_scheduler = [
    dict(
        type='LinearLR',
        start_factor=0.1,
        by_epoch=True,
        begin=0,
        end=5,
        convert_to_iter_based=True
    ),
    dict(
        type='CosineAnnealingLR',
        by_epoch=True,
        begin=5,
        T_max=25,                    # 총 에폭 - warmup 에폭
        eta_min=1e-6
    )
]
```

3.3 배치 크기 및 데이터로더

안정적인 설정

```
train_dataloader = dict(
    batch_size=32,                   # 배치 크기
    num_workers=8,                   # 워커 수
    persistent_workers=True,         # 지속적 워커
    sampler=dict(type='DefaultSampler', shuffle=True),
    dataset=dict(
        type=dataset_type,
        ann_file=ann_file_train,
        pipeline=train_pipeline,
        test_mode=False
    )
)
```

GPU 메모리별 배치 크기 권장

- **RTX 3090 (24GB)**: batch_size=32-48
- **RTX 4090 (24GB)**: batch_size=48-64
- **V100 (32GB)**: batch_size=64-96
- **A100 (40GB)**: batch_size=96-128

3.4 학습 설정

에폭 및 검증 설정

```
train_cfg = dict(
    type='EpochBasedTrainLoop',
    max_epochs=30,                   # 총 학습 에폭
    val_begin=1,                     # 검증 시작 에폭
    val_interval=2                   # 검증 간격
)
```

자동 학습률 스케일링

```
auto_scale_lr = dict(
    base_batch_size=128,              # 기준 배치 크기
    enable=True                       # 자동 스케일링 활성화
)
```

4. 평가 메트릭 및 모니터링

4.1 평가 메트릭

```
val_evaluator = [
    dict(
        type='AccMetric',
        metric_options=dict(
            top_k_accuracy=dict(topk=(1,)),      # Top-1 정확도
            mean_class_accuracy=dict()           # 클래스별 평균 정확도
        )
    )
]
```

4.2 체크포인트 및 로깅

```
default_hooks = dict(
    checkpoint=dict(
        type='CheckpointHook',
        interval=10,                  # 체크포인트 저장 간격
        save_best='auto',            # 최고 성능 모델 자동 저장
        max_keep_ckpts=5             # 최대 체크포인트 보관 수
    ),
    logger=dict(
        type='LoggerHook',
        interval=20,                 # 로그 출력 간격
        ignore_last=False
    )
)
```

4.3 시각화 백엔드

```
vis_backends = [
    dict(type='LocalVisBackend'),         # 로컬 저장
    dict(type='TensorboardVisBackend')    # TensorBoard 연동
]
visualizer = dict(
    type='ActionVisualizer',
    vis_backends=vis_backends
)
```

5. 학습 실행 가이드

5.1 데이터 준비

Stage1-3 실행하여 학습 데이터 생성

```
# Stage1: 포즈 추정
docker exec mmlabs bash -c "cd /workspace/recognizer && python3 main.py --mode annotation.stage1"

# Stage2: 추적
docker exec mmlabs bash -c "cd /workspace/recognizer && python3 main.py --mode annotation.stage2"

# Stage3: 데이터셋 생성
docker exec mmlabs bash -c "cd /workspace/recognizer && python3 main.py --mode annotation.stage3"
```

5.2 STGCN++ 학습 실행

Fight 감지 모델 학습

```
cd /workspace/mmaction2

# 단일 GPU 학습
python tools/train.py configs/skeleton/stgcnpp/stgcnpp_enhanced_fight_detection_stable.py

# 다중 GPU 학습 (4 GPUs)
bash tools/dist_train.sh configs/skeleton/stgcnpp/stgcnpp_enhanced_fight_detection_stable.py 4
```

Falldown 감지 모델 학습

```
cd /workspace/mmaction2

# 학습 실행
python tools/train.py configs/skeleton/stgcnpp/stgcnpp_enhanced_falldown_detection_stable.py
```

5.3 모델 테스트 및 평가

```
# 모델 테스트
python tools/test.py \
    configs/skeleton/stgcnpp/stgcnpp_enhanced_fight_detection_stable.py \
    work_dirs/stgcnpp-bone-ntu60_rtmo-l_RWF2000plus_stable/best_acc_top1_epoch_30.pth

# 다중 GPU 테스트
bash tools/dist_test.sh \
    configs/skeleton/stgcnpp/stgcnpp_enhanced_fight_detection_stable.py \
    work_dirs/stgcnpp-bone-ntu60_rtmo-l_RWF2000plus_stable/best_acc_top1_epoch_30.pth \
    4
```

6. 하이퍼파라미터 튜닝 가이드

6.1 모델 아키텍처 튜닝

네트워크 깊이 조정

```
# 더 깊은 네트워크 (높은 정확도, 느린 속도)
backbone=dict(
    num_stages=6,                    # 4 → 6으로 증가
    base_channels=64,
    inflate_stages=[2, 4],           # 다중 inflate 스테이지
    down_stages=[2, 4],
)

# 더 얕은 네트워크 (빠른 속도, 낮은 정확도)
backbone=dict(
    num_stages=3,                    # 4 → 3으로 감소
    base_channels=48,                # 채널 수 감소
    inflate_stages=[],               # inflate 없음
    down_stages=[],
)
```

채널 수 조정

```
# 높은 표현력 (더 많은 파라미터)
backbone=dict(
    base_channels=96,                # 64 → 96으로 증가
    # ...
)

# 경량화 (적은 파라미터)
backbone=dict(
    base_channels=32,                # 64 → 32로 감소
    # ...
)
```

6.2 정규화 기법

드롭아웃 조정

```
cls_head=dict(
    dropout=0.5,                     # 과적합 방지 (높은 값)
    # 또는
    dropout=0.1,                     # 언더피팅 방지 (낮은 값)
)
```

가중치 감쇠 조정

```
optimizer=dict(
    weight_decay=0.01,               # 강한 정규화
    # 또는
    weight_decay=0.0001,             # 약한 정규화
)
```

6.3 데이터 증강

시간적 증강

```
train_pipeline = [
    dict(type='PreNormalize2D'),
    dict(type='GenSkeFeat', dataset='coco', feats=['b']),
    dict(type='UniformSampleFrames', clip_len=100),
    dict(type='PoseDecode'),
    # 시간적 노이즈 추가
    dict(type='RandomRescale', scale_factor=(0.8, 1.2)),
    dict(type='FormatGCNInput', num_person=4),
    dict(type='PackActionInputs')
]
```

공간적 증강

```
train_pipeline = [
    dict(type='PreNormalize2D'),
    # 공간적 변환 추가
    dict(type='RandomShift', shift_ratio=0.1),
    dict(type='RandomRotate', angle=15),
    dict(type='GenSkeFeat', dataset='coco', feats=['b']),
    # ...
]
```

7. 성능 최적화 및 디버깅

7.1 학습 속도 최적화

혼합 정밀도 학습

```
# config 파일에 추가
fp16 = dict(loss_scale=512.0)
```

DataLoader 최적화

```
train_dataloader = dict(
    num_workers=16,                  # CPU 코어 수에 맞게 조정
    persistent_workers=True,         # 워커 재사용
    pin_memory=True,                 # GPU 메모리 고정
    prefetch_factor=4,               # 미리 가져올 배치 수
)
```

7.2 메모리 최적화

그래디언트 체크포인팅

```
model = dict(
    backbone=dict(
        # ...
        with_cp=True,                # 체크포인트 활성화
    )
)
```

그래디언트 누적

```
# 배치 크기를 줄이고 누적 단계를 늘림
optim_wrapper = dict(
    accumulative_counts=4,           # 4번 누적 후 업데이트
    # ...
)
```

7.3 학습 모니터링

조기 종료

```
default_hooks = dict(
    # ...
    early_stopping=dict(
        type='EarlyStoppingHook',
        monitor='val/acc_top1',
        patience=10,
        min_delta=0.001
    )
)
```

학습률 감시

```
default_hooks = dict(
    # ...
    lr_scheduler=dict(
        type='LrSchedulerHook',
        by_epoch=True
    )
)
```

8. 일반적인 문제 해결

8.1 학습이 안 되는 경우

학습률 조정

```
# 학습률이 너무 높은 경우
optimizer=dict(lr=0.00001)  # 1e-5로 감소

# 학습률이 너무 낮은 경우
optimizer=dict(lr=0.001)    # 1e-3으로 증가
```

배치 크기 조정

```
# 배치 크기 증가 (안정적 학습)
train_dataloader = dict(batch_size=64)

# 배치 크기 감소 (메모리 부족시)
train_dataloader = dict(batch_size=16)
```

8.2 과적합/언더피팅 해결

과적합 해결

```
# 더 강한 정규화
cls_head=dict(dropout=0.7)
optimizer=dict(weight_decay=0.01)

# 데이터 증강 추가
# (위의 데이터 증강 섹션 참조)
```

언더피팅 해결

```
# 모델 복잡도 증가
backbone=dict(
    num_stages=6,
    base_channels=96
)

# 정규화 감소
cls_head=dict(dropout=0.1)
optimizer=dict(weight_decay=0.0001)
```

8.3 GPU 메모리 부족 해결

```
# 배치 크기 감소
train_dataloader = dict(batch_size=8)

# 그래디언트 체크포인팅 활성화
model = dict(backbone=dict(with_cp=True))

# 혼합 정밀도 학습
fp16 = dict(loss_scale=512.0)
```