AI Deadlift Judge

Automated Detection of Technical Rule Violations in Powerlifting

Majid Alredha (2211329)

School of Computer Science

College of Engineering and Physical Sciences

University of Birmingham

2024-25

Abstract

Manual judging in powerlifting competitions often suffers from subjectivity and inconsistency, which can undermine fairness and lead to disputed results. The **AI Deadlift Judge** project addresses this by introducing a system that uses modern computer vision and machine learning techniques to detect technical rule violations in deadlift attempts.

The system uses a YOLO11 pose estimation model that extracts 2D joint key points from each video frame, capturing the athlete's movement. These key points are standardized through a data preprocessing pipeline to enhance the model's resilience to video variability.

This data is fed into a custom-designed TCN, which distills temporal features before outputting predictions for three distinct infraction cards as defined by the IPF. The training process, featuring balanced data sampling and a specialized loss function, has enabled the model to achieve robust performance despite challenges such as class imbalances and varying video quality.

Quantitative evaluation shows the effectiveness of the system with an accuracy of 80%, precision of 74.18%, recall of 92.31%, an F1 score of 82.15%, and a custom judge score of 83.33%. These metrics underscore the system's potential to deliver reliable assessments in competitive settings.

By automating the detection of infractions, the **AI Deadlift Judge** exemplifies how advanced data processing, model training, and real-time user interaction can merge into a practical solution to improve fairness and objectivity in sports judging.

Abbreviations

AI	Artificial Intelligence
BCE	Binary Cross Entropy
CNN	Convolutional Neural Network
CSV	Comma-Separated Value
GPU	Graphics Processing Unit
IPF	International Powerlifting Federation
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Network
TCN	Temporal Convolutional Network
YOLO	You Only Look Once (Object Detection System)

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Competitive powerlifting demands precision and fairness in judging, yet manual judging is inherently subjective and prone to inconsistency. Variability in human perception and judgment can lead to disputes, affect competition outcomes, and undermine confidence in the sport. The **AI Deadlift Judge** project was conceived to tackle these challenges by developing an automated, data-driven, and efficient system to evaluate deadlift attempts.

1.1 Background and Motivation

Traditional deadlift judging relies on the expertise and intuition of human judges, whose assessments may vary due to differences in experience, fatigue, or even momentary lapses in attention, challenges that are common during long back-to-back judging sessions. This project leverages recent advances in computer vision and machine learning to address these issues.

1.2 Problem Definition

The central challenge addressed by the **AI Deadlift Judge** is the reliable and automated detection of deadlift infractions as defined by the IPF rulebook.

DEADLIFT INFRACTION CARDS

RED CARD	BLUE CARD	YELLOW CARD
Failure to lock the knees straight at the completion of the lift. Failure to stand erect with the shoulders back.	Any downward movement of the bar before it reaches the final position. If the bar settles as the shoulders come back this should not be reason to disqualify the lift. Supporting the bar on the thighs during the performance of the lift. If the bar edges up the thighs but is not supported, this is not reason for disqualification.	Lowering the bar before receiving the Chief Referees signal. Allowing the bar to return to the platform without maintaining control with both hands. Failure to comply with any of the requirements contained in the general description of the lift, which precedes this list of disqualification. Incomplete lift

Figure 1.1: IPF Rulebook - Deadlift Infraction Cards

Deadlift

1. The lifter shall face the front of the platform with the bar laid horizontally in front of the lifter's feet, gripped with an optional grip in both hands and lifted until the lifter is standing erect.
2. On completion of the lift the knees shall be locked in a straight position and the shoulders back.
3. The Chief Referee's signal shall consist of a downward movement of the arm and the audible command "Down". The signal will not be given until the bar is held motionless and the lifter is in the apparent finished position.
4. Any rising of the bar or any deliberate attempt to do so will count as an attempt. Once the attempt has begun no downward movement is allowed until the lifter reaches the erect position with the knees locked. If the bar settles as the shoulders come back (slightly downward on completion) this should not be reason to disqualify the lift.

Figure 1.2: IPF Rulebook - Deadlift Rules of Performance

This requires a system capable of:

- **Robust Pose Estimation:** Accurately extracting normalized joint key points from video frames despite varying lighting, camera angles, and occlusions.
- **Temporal Sequence Analysis:** Employing a Temporal Convolutional Network (TCN) to interpret dynamic movement over a standardized

sequence of 200 frames.

- **Handling Data Variability:** Mitigating challenges such as class imbalances and inconsistencies in video quality using sophisticated preprocessing and balanced training techniques.

1.3 Objectives and Scope

The primary aims of this project include:

- **Developing an AI-driven Evaluation Pipeline:** Creating a robust system that combines YOLO11-based pose estimation with a TCN for temporal analysis, thereby automating the detection of deadlift infractions.
- **Creating a High-Quality Training Database:** Curating a comprehensive and consistent training database of deadlift attempt videos, with labels corresponding to technical violation codes.
- **Enhancing Objectivity and Consistency:** Aiding subjective human judgment with a data-driven approach that can consistently identify rule violations.
- **Real-World Applicability:** Demonstrating the system's utility through an interactive, user-friendly interface built with Streamlit for real-time video processing and feedback.
- **Rigorous Evaluation:** Validating the system using both quantitative metrics and qualitative analyses from live demonstrations.

The overarching goal of this project is to enhance fairness and accuracy in powerlifting competitions. By integrating advanced AI methodologies with a user-oriented design, the **AI Deadlift Judge** aims to pave the way for future innovations in objective performance evaluation.

CHAPTER 2

Literature Review / Related Work

The rapid evolution of computer vision and deep learning has led to transformative developments in sports analytics. Many studies have explored human action recognition and pose estimation in dynamic environments, yet none have focused on using these techniques to detect specific technical rule violations in sports such as powerlifting.

Studies in judged sports demonstrate biases arising from judge subjectivity, highlighting the potential for systematic error in competition outcomes [9].

2.1 Existing Systems and Techniques

Early work in object detection and pose estimation, notably through the YOLO framework developed by Redmon et al. [13], revolutionized real-time analysis by providing rapid and accurate detection capabilities. Building on this foundation, later adaptations have extended YOLO for pose estimation tasks, extracting normalized key points that accurately represent human joints. In this project, a YOLO11 model is used, drawing inspiration from both YOLO's detection efficiencies and methods like Part Affinity Fields [2]. Alternative pose estimation frameworks, such as Simple Baselines [17], provide complementary design insights for keypoint head architectures.

Deep convolutional networks have seen significant improvements through architectural innovations such as residual learning, as introduced by He et al. [4]. Their work on Deep Residual Networks (ResNets) addressed the challenges of training very deep models by incorporating skip connections, a concept that has influenced many modern architectures. Alongside this, foundational insights provided by LeCun, Bengio, and Hinton [7] have established deep learning as a transformative approach for feature extraction and pattern recognition. Although our approach centers on a YOLO-based pose estimator

and a TCN for sequence modeling, these underlying principles have informed our design choices by emphasizing stability and improved feature extraction.

In the realm of temporal sequence analysis, traditional RNNs and LSTM models have been widely used. However, recent research highlights the advantages of TCNs in capturing long-range dependencies with enhanced training stability and efficiency [1]. Lea et al. demonstrate that TCNs excel at action segmentation and detection by leveraging dilated convolutions to capture extensive temporal context [6]. An empirical comparison by Bai et al. further confirms TCNs' superiority over both convolutional and recurrent architectures for sequence modeling tasks [1].

Data augmentation techniques, such as horizontal flipping and geometric transformations, help mitigate overfitting and improve generalization [14]. In our pipeline, we employ horizontal flipping of normalized keypoints to effectively double the dataset size without altering semantic information.

2.2 Comparison to Related Work

While numerous automated systems have been proposed for sports performance analysis, most have concentrated on general action recognition or overall performance enhancement rather than enforcing technical rules. Traditional methods, including CNN-based and hybrid CNN-RNN approaches, have been applied to classify movements in sports; however, they often lack the specificity needed for judging technical infractions. AI Deadlift Judge is unique in offering a rule-based evaluation mechanism that directly supports competition judging.

Additionally, the integration of pose estimation with TCN-based temporal analysis is a significant methodological advancement. Unlike systems that separate spatial and temporal analyses, our unified pipeline uses consistent preprocessing, such as fixed-length sequence standardization and augmentation, to address practical challenges like video variability and class imbalances. This integrated approach not only improves detection accuracy but also enhances robustness under real-world conditions.

2.3 Gap Analysis

A thorough review of existing literature and systems reveals several key gaps that the AI Deadlift Judge project addresses:

- **Specificity in Infraction Detection:** While earlier research has focused on general activity recognition, no systems translate pose and motion data directly into assessments of rule violations. By training the model on movement patterns linked to IPF-defined infractions, AI Deadlift Judge can offer actionable insights for competitive judging.
- **Integrated Spatio-Temporal Analysis:** Many systems treat pose extraction and temporal modeling as separate tasks. Our approach

combines a YOLO-based pose estimator with a TCN that processes sequential data in a single, cohesive framework, effectively capturing both spatial and temporal dynamics.

- **Robust Handling of Data Variability:** Real-world sports footage is often affected by factors such as lighting variations, occlusions, and camera angle differences. Although some studies acknowledge these challenges, few incorporate comprehensive preprocessing steps like sequence padding and data augmentation to systematically mitigate these issues.
- **Balancing Class Imbalances:** In sports analytics, certain infraction types may be underrepresented in training datasets. Unlike many existing approaches, our method uses a `WeightedRandomSampler` and adjusts loss functions with positive weight factors to ensure that infrequent infractions are effectively learned.

To further mitigate class imbalance, we also adopt the Class–Balanced Loss based on the effective number of samples, as proposed by Cui et al. [3].

In summary, while the fields of pose estimation and sequence modeling are well established, applying these techniques to the domain of deadlift judging is a novel contribution. By addressing both technical challenges and real-world variability, AI Deadlift Judge advances automated sports analysis and sets a new benchmark for objective performance evaluation.

CHAPTER 3

Methodology Overview

This chapter details the strategies used in the architecture, dataset creation, data handling, modelling, and evaluation of the project, along with justifications behind the decisions made.

3.1 System Architecture

3.1.1 Video Processing & Pose Estimation

The system begins with video input from competitive deadlift attempts. A pretrained YOLO11 model is used to extract 2D joint key points from each frame. Each frame is processed to detect 17 normalized joint positions, producing a sequence of key points that encapsulate the athlete's position. This extraction leverages the real-time capabilities of YOLO-based models as described by Redmon et al. [13].

3.1.2 Data Preprocessing & Standardization

Raw key point data is often variable in length due to differences in video duration. To standardize the input, a dedicated padding function truncates or pads the sequence to exactly 200 frames. The processed tensor is then reshaped from (200, 17, 2) into a 2D tensor of shape (34, 200), where 34 represents the flattened joint coordinates (17 joints $\times$ 2 coordinates). Data augmentation, in the form of horizontal flipping, is applied during preprocessing to effectively double the dataset and improve model generalization with variations in camera angles.

```
def pad_sequence(tensor, max_length=200):
    # Pad or truncate the input tensor along the first dimension to a fixed length
    num_frames = tensor.shape[0]
    if num_frames < max_length:
        padding = torch.zeros(max_length - num_frames, tensor.shape[1], tensor.shape[2], device=tensor.device)
        return torch.cat([tensor, padding], dim=0)
    return tensor[:max_length]
```

Figure 3.1: Data Standardization - Pad Sequence Function

```
def horizontal_flip(stacked_tensor):
    # Returns a new tensor where x-coordinates values are flipped (1 - x) for nonzero values
    flipped = stacked_tensor.clone() # Clone the input tensor
    x = flipped[..., 0]  # Extract the x-coordinate values Shape: (N, 200, 17)
    mask = x != 0 # Create a mask when x is nonzero
    x[mask] = 1 - x[mask] # Apply the flip where the mask is True
    flipped[..., 0] = x # Update the flipped tensor's x-coordinates
    return flipped
```

Figure 3.2: Data Augmentation - Horizontal Flip Function

3.1.3 Temporal Analysis with TCN

The core analytical component is a custom-designed TCN implemented using PyTorch and the `pytorch_tcn` library. The network is structured with five convolutional layers with channels [50, 75, 100, 125, 125]. The lower layers capture simple, local temporal patterns, while higher layers extract more complex and abstract features. Increasing the number of filters as you go deeper allows the network to learn a richer set of features. This progression is a common design strategy in convolutional architectures to balance computational efficiency with representational power.

The network also uses a kernel size of 7, which means that each convolution will consider 7 consecutive time steps, which is a compromise between capturing short-term dynamics and incorporating enough contextual information over the 200-frame sequence and led to the best quantitative results when evaluated against kernel sizes of 3, 5, and 9.

A dropout rate of 0.5 was also used. This is used as a regularization technique to prevent overfitting by randomly deactivating half of the neurons

during each forward pass of a batch in training [15]. A dropout rate of 0.5 is common in scenarios where the dataset is limited or imbalanced, as it ensures that the model does not become overly reliant on any specific set of features and generalizes better to unseen data.

After processing the input sequence, global average pooling [8] condenses the temporal features before passing them through a fully connected layer that outputs raw logits for the three infraction types. The design decisions in the TCN are supported by literature demonstrating the effectiveness of convolutional architectures in sequence modeling [1].

3.1.4 Output & User Interface

The final stage converts the TCN's raw logits into binary predictions via sigmoid activation. These predictions are then interpreted into a human-readable format (e.g., "Good Lift" or "No Lift") and displayed through a Streamlit-based interface, ensuring real-time feedback during demonstrations. This integration of computer vision, temporal analysis, and user interactivity exemplifies a seamless and practical solution for automated deadlift judging.

3.2 Dataset and Labelling

3.2.1 Manually Created Training Dataset

A distinctive feature of this project is the manually curated training dataset. Videos of deadlift attempts were recorded and meticulously selected to cover a diverse range of scenarios, including varying lighting conditions, different camera angles, and diverse athlete profiles. Each video in the dataset was manually annotated with binary labels corresponding to three key infraction types. This direct curation ensured high-quality, accurate labels, which in turn improved the TCN's ability to learn subtle differences between compliant lifts and technical violations.

The following table summarizes the breakdown of the entire dataset, after augmentation is applied, by infraction category:

Category	**Count**
Total videos	750
Red card	60
Blue card	52
Yellow card	238
Good lifts	418

Table 3.1: Composition of the Deadlift Video Dataset.

3.2.2 Labeling and Handling Imbalances

Ground truth labels for infractions are loaded from a CSV file. Each video is annotated with binary labels corresponding to three infraction types. To address class imbalances, a common issue in sports analytics, the training pipeline employs a WeightedRandomSampler, which attempts to draw samples more evenly across the 3 infraction types. It also computes positive weight adjustments for the `BCEWithLogitsLoss` function, which increases the penalty for misclassifying a positive example when positive samples are rare. This ensures that less frequent infractions are given adequate importance during training.

```
def compute_pos_weight(labels_csv_path):
    # Compute pos_weight for BCEWithLogitsLoss from the CSV file
    df = pd.read_csv(labels_csv_path)
    pos_weight_list = []
    for col in ['Red', 'Blue', 'Yellow']:
        pos_count = df[col].sum()
        neg_count = len(df) - pos_count
        ratio = neg_count / pos_count if pos_count > 0 else 1.0
        pos_weight_list.append(ratio)
    return torch.tensor(pos_weight_list, dtype=torch.float32)
```

Figure 3.3: Model Training - Positive Weights Function

3.3 Model Training and Evaluation

3.3.1 Training Setup

A DataLoader is created from the dataset, splitting it into training and validation subsets. The DataLoader is used to efficiently load data in mini-batches, which reduces memory overhead and speeds up the training process. It provides a consistent mechanism for shuffling and sampling the data, ensuring that the model sees a diverse range of samples during each epoch.

The model is trained using the Adam optimizer with a learning rate of 0.001 [5]. This optimizer was selected because it offers adaptive learning rates, where each parameter is updated based on the first and second moment estimates of the gradients, which is particularly beneficial for managing the noisy temporal data processed by the TCN. This adaptability also ensures reliability and efficiency to accelerate convergence.

A moderate batch size of 64 was selected to provide a stable estimate of the gradients while balancing computational load and memory constraints. An extended training period of 400 epochs was chosen to ensure that the custom

TCN had ample opportunities to learn the complex temporal patterns present in the data. This allows the network to repeatedly refine its weights, which is particularly important given the subtle movement nuances in the dataset.

The training process employs a BCE loss with logits, which is specifically designed for binary classification tasks. It combines the sigmoid activation and BCE loss into one single function, which improves numerical stability compared to applying these steps separately. This stability is vital when dealing with raw logits, as it helps prevent issues like vanishing gradients. This loss function is enhanced with the positive weights tensor to balance the training samples across the three infraction classes.

3.3.2 Hyperparameter Tuning

Extensive experimentation with hyperparameters, such as the number of convolutional filters, kernel size, dropout rate, and sequence length, was conducted to stabilize the training process and optimize performance. This tuning process was crucial in achieving the reported evaluation metrics.

Table 3.2: Hyperparameter Summary

Hyperparameter	**Value**
Learning rate	0.001
Batch size	64
Sequence length	200 frames
Kernel size	7
Dropout rate	0.5
Number of TCN channels	[50, 75, 100, 125, 125]
Number of training epochs	400
Optimizer	Adam
Classification threshold	0.9

3.3.3 Evaluation Metrics

In addition to standard metrics of accuracy, precision, recall, and F1 score, a custom judge score is calculated to reflect the system's real-world efficacy in deadlift judging.

The judge score is a custom metric that measures how often the system's overall decision aligns with human judgment. In this context, a "good lift" is defined as a lift where none of the three infractions are detected, while a "no lift" is one in which at least one infraction is flagged. This method provides an easily interpretable performance measure by directly reflecting the real-world criteria used by IPF judges, ensuring that the system's predictions are practical and meaningful in the context of actual deadlift judging.

```
    def judge_score(labels, predictions):
            # For each sample, sum the three values.
            pred_goodlift = (predictions.sum(dim=1) == 0)
                # If the sum is 0, it's a goodlift
            true_goodlift = (labels.sum(dim=1) == 0)
            misclassified = (pred_goodlift !=
                true_goodlift).float().mean().item() #
                misclassification when the prediction doesn
                't match the label.
            return (1.0 - misclassified)
```

Figure 3.4: Model Evaluation - Custom Judge Score

3.4 Integration and Real-World Demonstration

3.4.1 Integration with Streamlit

The complete pipeline, from video upload and pose extraction to real-time prediction, is integrated into a user-friendly application. This interface is designed to facilitate live demonstrations, allowing judges and athletes to interact with the system seamlessly.

3.4.2 Iterative Development

Throughout the project, iterative development and continuous testing have been paramount. Feedback from early demonstrations and quantitative evaluation guided refinements in data processing, model architecture, and user interaction design. This iterative cycle of development and testing ensured that the final system is both robust and applicable in competitive settings.

In summary, the AI Deadlift Judge combines modern pose estimation, effective data preprocessing and augmentation, and a carefully tuned TCN for temporal analysis, trained on a manually curated training dataset, along with a responsive user interface. This integrated approach, supported by contemporary literature on deep learning and sequence modeling, ensures that the system provides objective, consistent, and actionable evaluations of deadlift attempts.

CHAPTER 4

System Implementation

This chapter describes the structure and implementation of the AI Deadlift Judge system. The project is divided into three main components:

1. **Video Data Processing Notebook:** This module loads all raw deadlift videos from the dataset, extracts pose key points using the *YOLO11x-pose* model, processes and standardizes the data into a tensor format, applies horizontal flipping, and saves the final tensor to disk.

2. **Model Training and Evaluation Notebook:** In this component, the saved video tensor is loaded along with label data from a CSV file. A custom Dataset class is used to pair each video tensor with its corresponding binary labels for three infraction types. The custom TCN is defined and trained using this data. The training is executed with a batch size of 64 over 400 epochs, employing the Adam optimizer with a learning rate of 0.001 and BCE with logits loss augmented with positive weights. After extensive hyperparameter tuning and evaluation, the model weights are saved to disk.

3. **Streamlit Application:** The final component is a user interface built with Streamlit [16]. Here, the saved model weights are loaded, and a new video is processed with the *YOLO11n-pose* model, which is used as it reduces latency during live usage by speeding up the key point extraction step. The processed video tensor is fed through the TCN, and the output logits are converted into a human-readable format, indicating whether the lift is classified as a "Good Lift" or "No Lift" flagging all infraction cards that were detected.

4.1 Video Data Processing

The first component is implemented in a dedicated notebook, `LoadVideoData.ipynb`. Key steps include:

- **Pose Extraction:** For training data, the *YOLO11x-pose* model is employed to extract 17 normalized joint key points from each frame, ensuring high accuracy and data quality.
- **Sequence Standardization:** A dedicated function is used to standardize video lengths by truncating or padding each sequence to 200 frames.
- **Data Augmentation:** Horizontal flipping is applied to effectively double the dataset size and enhance model generalization.
- **Saving Data:** The resultant tensor, combining both original and augmented data, is saved to disk for use in the training phase using PyTorch [10].

```python
class Video:
    # Class to load and process a video given its ID
    def __init__(self, video_id):
        self.video_id = video_id
        self.data = self.getdata()

    def getdata(self):
        print(f"Loading video {self.video_id} ...")
        try:
            # Process the video file from the DLdataset directory
            results = posemodel(f"DLdataset/{self.video_id}.mp4", iou=0.4, verbose=False, max_det=1, stream=True)
        except Exception as e:
            print(f"Error loading video {self.video_id}: {e}")
            return None

        tensors_list = []
        for result in results:
            try:
                # Extract normalized keypoints (shape: [17, 2])
                keypoints = result.keypoints.xyn.reshape(17, 2)
                tensors_list.append(keypoints)
            except Exception as e:
                print(f"Error processing frame in video {self.video_id}: {e}")
        if not tensors_list:
            print(f"No valid frames found in video {self.video_id}.")
            return None

        video_tensor = torch.stack(tensors_list)  # Shape: [num_frames, 17, 2]
        video_tensor = pad_sequence(video_tensor, max_length=200)  # Now shape: [200, 17, 2]
        print(f"Video {self.video_id} loaded.")
        return video_tensor
```

Figure 4.1: Video Class [LoadVideoData.ipynb]

```python
def process_and_stack_videos(video_ids, output_file="all_videos.pt"):
    # Processes each video in video_ids using the Video class
    video_tensors = []
    for vid in video_ids:
        video_instance = Video(vid)
        data = video_instance.data
        if data is not None:
            video_tensors.append(data)
        else:
            print(f"Skipping video {vid} because it could not be processed.")

    if not video_tensors:
        raise RuntimeError("No videos were successfully processed.")

    # Stack the list of tensors into a single tensor
    stacked_tensor = torch.stack(video_tensors)  # Shape: (N, 200, 17, 2)
    flipped_tensor = horizontal_flip(stacked_tensor) # Shape: (N, 200, 17, 2)
    output_tensor = torch.cat((stacked_tensor, flipped_tensor), dim=0) # Shape: (N*2, 200, 17, 2)
    torch.save(output_tensor, output_file)
    print(f"Saved tensor with shape {output_tensor.shape} to {output_file}")
```

Figure 4.2: Process all Videos Function [LoadVideoData.ipynb]

4.2 Model Training and Evaluation

The second module, `DLJudgeTrain.ipynb,` is responsible for training the TCN. The process involves:

- **Data Loading:** The preprocessed tensor and corresponding CSV labels are loaded. The data is reshaped from size (200, 17, 2) to (34, 200) and a custom Dataset class then pairs each video tensor with its respective three-dimensional binary label vector. A DataLoader then splits the data into training and validation subsets and uses the WeightedRandomSampler to ensure balanced batch sampling during the training process [11].
- **TCN Architecture:** In the forward pass of the TCN, the input tensor is first processed through the TCN block, which applies the series of convolutional layers to extract temporal features over 7 consecutive time steps. Next, global average pooling is applied along the time dimension, condensing the feature maps into a fixed-size vector for each sample. Finally, a fully connected layer maps these features to raw logits corresponding to the three infraction classes.
- **Training Setup:** Using a batch size of 64 and training over 400 epochs. The Adam optimizer is utilized with a learning rate of 0.001, and BCEwithlogitsloss is augmented with positive weights to handle class imbalances effectively.
- **Evaluation and Saving Weights:** The model is evaluated using standard metrics and a custom judge score that reflects agreement with human judgements. The final model weights are then saved for deployment.

```python
class Video:
    # Class to load and process a video given its ID
    def __init__(self, video_id):
        self.video_id = video_id
        self.data = torch.load("all_videos.pt",
            map_location=device, weights_only=True)[
            video_id-1] # [200, 17, 2]
        self.data = self.data.view(200, -1).transpose(0,
            1) # -> (200, 34) -> (34, 200)

class DeadliftVideoDataset(Dataset):
    def __init__(self, video_ids, labels_csv_path,
        transform=None):
        self.video_ids = video_ids # List of video IDs
        self.labels_dict = get_labels_from_csv(video_ids,
            labels_csv_path)
        self.transform = transform # Optional transform to
             be applied to the video tensor

    def __len__(self):
        return len(self.video_ids)

    def __getitem__(self, idx):
        vid = self.video_ids[idx]
        video_instance = Video(vid)
        data = video_instance.data  # Expected shape:
            [200, 17, 2]
        if data is None:
            raise ValueError(f"Data for video {vid} could
                not be loaded.")
        if self.transform:
            data = self.transform(data)
        labels = self.labels_dict[vid]
        return data, labels, vid
```

Figure 4.3: Video & Dataset Classes [DLJudgeTrain.ipynb]

```
class DeadliftTCN(nn.Module):
    def __init__(self, input_size, num_channels, num_classes, kernel_size=2, dropout=0.2):
        super(DeadliftTCN, self).__init__()
        self.tcn = TCN(input_size, num_channels, kernel_size=kernel_size, dropout=dropout)
        self.fc = nn.Linear(num_channels[-1], num_classes)

    def forward(self, x): # x shape: (batch_size, input_size, sequence_length)
        y = self.tcn(x)  # Shape: (batch, num_channels[-1], sequence_length)
        y = torch.mean(y, dim=2)  # Global average pooling over time dimension
        logits = self.fc(y)       # Raw logits (BCEWithLogitsLoss expects raw logits)
        return logits

# Initialize model and hyperparameters
labels_csv_path = "DLdataset/simple_labels.csv"
video_ids = pd.read_csv(labels_csv_path)["ID"].tolist()
batch_size = 64
train_loader, val_loader, dataset = create_dataloaders(video_ids, labels_csv_path, batch_size=batch_size) # Create DataLoaders
pos_weight_tensor = compute_pos_weight(labels_csv_path).to(device) # Compute pos_weight for loss balancing
print(f"Pos weight: {pos_weight_tensor}")

# Model hyperparameters
input_size = 34 # 17 joints * 2 coordinates per frame, flattened
num_classes = 3 # 3 infractions
num_channels = [50,75,100,125,125] # Filters per layer
kernel_size = 7
dropout = 0.5

# Initialize model, loss function, and optimizer
model = DeadliftTCN(input_size, num_channels, num_classes, kernel_size, dropout).to(device)
criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight_tensor)
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Figure 4.4: Custom TCN Class & Initialization [DLJudgeTrain.ipynb]

4.3 Deployment and User Interface

The final component is the Streamlit application (e.g., `DLJudgeApp.py`) designed for real-time evaluation. Its key functions are:

- **Model Loading:** The app loads the saved model weights to ensure immediate inference without retraining.
- **Video Upload and Processing:** Users can upload a new video; the app then processes it using the *YOLO11n-pose* model which is utilised to reduce latency during live demonstrations.
- **Inference and Output:** Once processed, the video tensor is fed into the TCN to produce raw logits. The logits are then activated via a sigmoid function and converted into human-readable outputs: a "Good Lift" if no infractions are detected, or "No Lift" if one or more infractions are present.

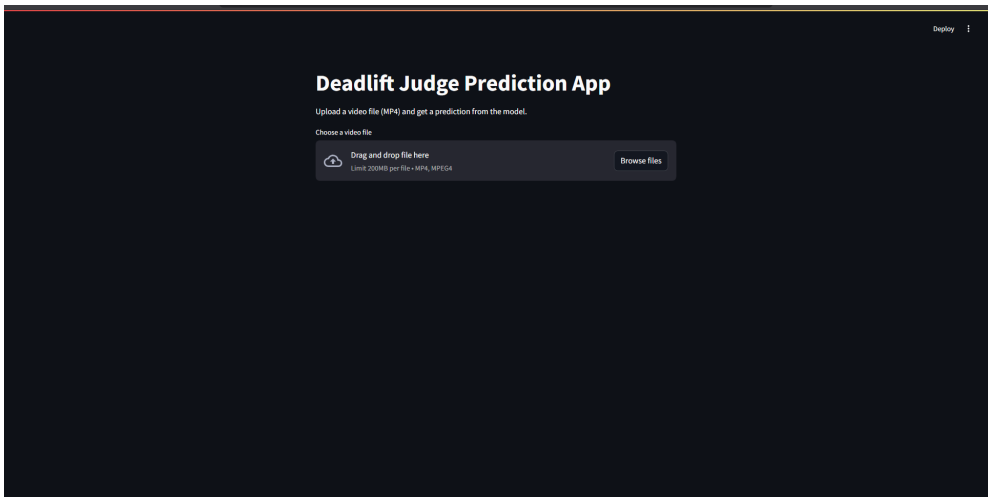

Figure 4.5: UI - Uploading a Video

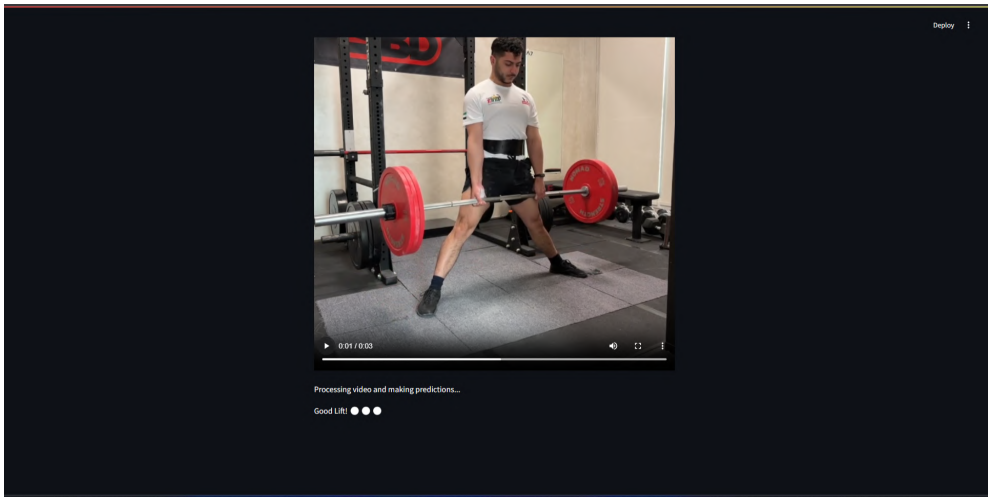

Figure 4.6: UI - Good Lift Output

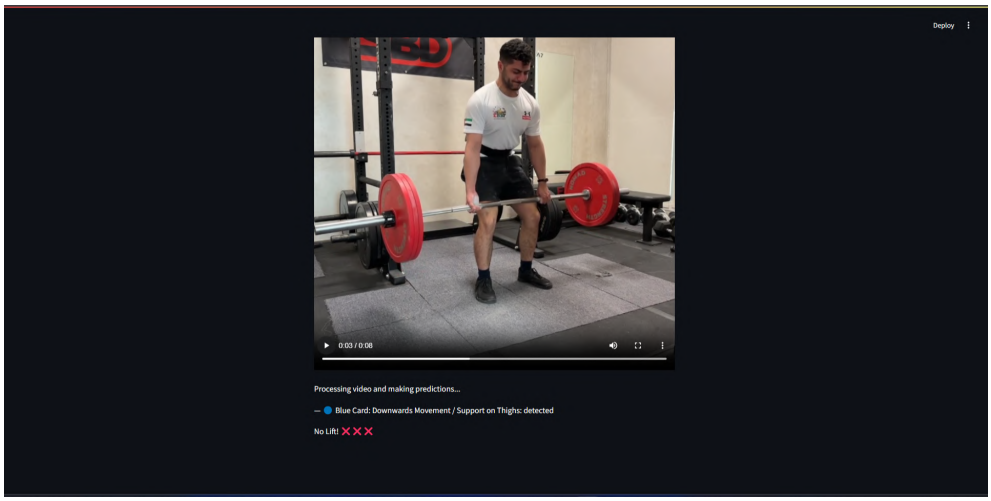

Figure 4.7: UI - No Lift (Blue Card) Output

4.4 Integration and Modular Workflow

The system reflects a modular and integrated approach which is designed to work in a fully automated manner. When a video is uploaded by the user, it is immediately processed: key points are extracted, sequences are standardized, predictions are made by the TCN, and the results are displayed in a user-friendly format. The system also automatically detects whether a GPU is available (via CUDA), ensuring optimal performance during both training and practical use.

- The **Data Processing** module ensures that raw video data is converted into a standardized tensor format using the *YOLO11x-pose* model.

- The **Training** module uses this tensor and manually curated labels to train a robust TCN, saving the model weights for later use.
- The **Deployment** module, implemented via a Streamlit app, leverages the speed of the *YOLO11n-pose* model to provide immediate predictions.

This separation allows independent development and testing of each stage, enhancing maintainability and supporting iterative refinement without cross-component interference.

CHAPTER 5

Results and Evaluation

The performance of AI Deadlift Judge was rigorously evaluated using both quantitative metrics and qualitative assessments. This evaluation not only measured the system's technical performance but also examined its practical applicability in real-world scenarios.

5.1 Quantitative Evaluation

The system's performance while using the *YOLO11x-pose* model was measured over a validation set using standard classification metrics [12] along with a custom judge score that reflects its practical utility:

- **Accuracy (80.00%):** Shows that 80% of all predictions, whether identifying an infraction or a "Good Lift" were correct.
- **Precision (74.18%):** Reflects the system's ability to minimize false positives. A precision of 74.18% shows that when the model flags an infraction, it is usually correct.
- **Recall (92.31%):** Measures the model's capability to capture actual infractions. A high recall of 92.31% means that the system rarely misses true violations.
- **F1 Score (82.15%):** The harmonic mean of precision and recall, indicating a balanced performance.
- **Judge Score (83.33%):** A custom metric that assesses the agreement between automated predictions and manual judgments. This score confirms that the system's outputs align well with the results of human judging.

Metric	YOLO11x-pose	YOLO11n-pose
Accuracy (%)	80.00	76.67
Precision (%)	74.18	77.95
Recall (%)	92.31	76.92
F1 Score (%)	82.15	76.78
Judge Score (%)	83.33	80.00

Table 5.1: Evaluation metrics using the YOLO11x and YOLO11n models.

Evaluation times were measured within the Streamlit application from the moment the user uploads a video to the moment a result is returned. Both YOLO11x-pose and YOLO11n-pose models were tested under identical conditions on an Nvidia RTX 4060 GPU, as shown in the table below:

Time (s)	YOLO11x-pose	YOLO11n-pose
Average	5.67	2.33
Maximum	9.91	3.78

Table 5.2: Evaluation times using the YOLO11x & YOLO11n models.

These figures reveal a trade-off between detection speed and performance. The YOLO11n-pose model, optimized for faster inference, misses more true infractions, resulting in slightly lower overall accuracy and F1 scores, a consequence expected when speed is prioritized. However, a custom judge score of 80.00% indicates that the system still maintains a good level of agreement with manual judgments. This trade-off justifies the adoption of the YOLO11n-pose model in scenarios where real-time performance is critical.

```python
def evaluate_model(model, data_loader, device):
    # Evaluate the model on the given data_loader and return metrics
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for inputs, labels, vids in data_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            # Apply sigmoid to convert logits to probabilities
            probs = torch.sigmoid(outputs)
            preds = (probs > 0.9).float()
            all_preds.append(preds.cpu())
            all_labels.append(labels.cpu())
    all_preds = torch.cat(all_preds)
    all_labels = torch.cat(all_labels)
    acc = accuracy_score(all_labels, all_preds)
    precision = precision_score(all_labels, all_preds, average='weighted', zero_division=0)
    recall = recall_score(all_labels, all_preds, average='weighted', zero_division=0)
    f1 = f1_score(all_labels, all_preds, average='weighted', zero_division=0)
    judge = judge_score(all_labels, all_preds)
    print(f"Evaluation Metrics -> Acc: {acc:.4f} | Precision: {precision:.4f} | Recall: {recall:.4f} | F1: {f1:.4f}")
    print(f"Evaluation Metrics -> Judge Score: {judge:.4f}")
```

Figure 5.1: Model Evaluation Function

5.2 Qualitative Evaluation

In addition to numerical performance, the system was evaluated qualitatively using a diverse set of video samples:

- **Robustness Across Conditions:** Videos featuring varied lighting, camera angles, and resolutions were processed. The system reliably detected clear infractions, matching well with manual annotations. This consistency is particularly significant for demonstrating the real-world applicability of the system.
- **Examples of Accurate Predictions:** In most instances, the system accurately flagged soft lockouts, improper downward movement, and incomplete lifts, reinforcing the strength of the pose estimation and tem-

poral analysis pipeline.

- **Observed Misclassifications:** Occasional misclassifications were noted, particularly in videos with partial occlusions or sudden changes in camera angle. These instances highlight the sensitivity of key point extraction to video quality and suggest that further improvements in preprocessing and augmentation may enhance robustness.

Figure 5.2: Correctly classified videos from various angles

5.3 Discussion of Challenges and Limitations

While the overall performance is promising, several challenges appeared during evaluation:

- **Data Imbalances:** Despite employing a WeightedRandomSampler and adjusting the `BCEWithLogitsLoss` with a `pos_weight` tensor, variations

in the frequency of different infractions still pose a challenge. Rare infraction types are less represented, which may affect the precision and recall for those classes.

- **Video Quality and Key Point Extraction:** Variability in video quality, such as poor lighting, occlusions, or unusual camera angles, can degrade the performance of the YOLO11 pose estimator, leading to less reliable key point extraction. This, in turn, impacts the subsequent temporal analysis.
- **Temporal Variability:** Rapid transitions or subtle movement nuances in deadlift attempts occasionally challenge the TCN's ability to capture critical temporal patterns. Fine-tuning the network architecture or incorporating additional temporal features may help mitigate these issues.

5.4 Comparison with Manual Judging

Preliminary comparisons between the automated assessments and manual judgments reveal several key advantages:

- **Enhanced Consistency:** Automated analysis provides repeatable and consistent evaluations, reducing the variability inherent in human judgment.
- **Improved Objectivity:** Decisions based on measurable key point data and temporal patterns offer a more impartial evaluation compared to subjective human observations.
- **Supportive Role:** While AI Deadlift Judge is not intended to fully replace human judges, it serves as a valuable support tool. By highlighting potential infractions for further review, the system helps judges make more informed decisions.

It is important to note that even a "perfect model" will not align perfectly with human judges since no two judges interpret the rulebook in exactly the same way. Subtle differences can lead to occasional discrepancies, so a flawless automated system will still diverge slightly from the decisions of any given panel.

5.5 Summary

In summary, the AI Deadlift Judge system shows promising performance, although challenges with data imbalances and video quality remain.

These findings validate the integration of advanced pose estimation and temporal analysis techniques while also illuminating areas for further improvement, thus laying a solid foundation for future improvements in automated sports judging.

CHAPTER 6

Discussion

By integrating modern techniques in pose estimation and temporal modeling, the system offers a robust alternative to traditional human judging. However, as with any pioneering system, there are both notable strengths and challenges that call for discussion.

6.1 Strengths and Innovations

6.1.1 Objective Evaluation

The system's ability to provide repeatable and consistent assessments addresses a long-standing issue in manual judging. The AI Deadlift Judge minimizes the variability inherent in human judgment. This is a crucial step forward in promoting fairness and transparency in competitive powerlifting.

6.1.2 Manually Created Training Dataset

A distinctive aspect of the project is the manual creation of the training dataset used to train the TCN. The manual curation ensured high-quality labels and enabled the system to focus on critical movements, enhancing its ability to distinguish between compliant lifts and violations.

6.1.3 Robust Preprocessing

Standardizing video sequences to a fixed length and incorporating data augmentation techniques such as horizontal flipping have been instrumental in improving the model's robustness. These preprocessing steps help mitigate issues related to variable video quality and camera angles, which are common in real-world scenarios.

6.2 Challenges and Limitations

6.2.1 Data Imbalance

Despite using a WeightedRandomSampler and adjusting loss functions with positive weight factors, class imbalances are still a challenge. Certain infraction types are inherently underrepresented in the dataset, which can affect the model's ability to learn about these rarer events. Future work might explore advanced augmentation strategies or synthetic data generation to address these imbalances more effectively.

6.2.2 Variability in Video Quality

The performance of the YOLO11 pose estimator is highly dependent on video quality. Poor lighting, occlusions, and unusual camera angles can lead to inaccurate key point extraction, which in turn affects the later temporal analysis. Enhancing the preprocessing pipeline to include adaptive quality checks and error correction mechanisms may help alleviate this issue.

6.2.3 Temporal Dynamics and Subtle Movements

Although the TCN is effective in capturing temporal patterns, certain rapid transitions or subtle nuances in deadlift attempts can still be challenging to model. Fine-tuning the network architecture or integrating additional temporal features, such as acceleration or joint angle dynamics, could potentially improve the system's sensitivity to these movements.

6.3 Future Directions

6.3.1 Dataset Expansion and Enrichment

While the current manually curated dataset has been invaluable in training the TCN, expanding the dataset to include more varied scenarios and infraction types would enhance the model's generalizability. Incorporating videos from different competitions and diverse athlete profiles can further bolster the system's robustness.

6.3.2 Hybrid Modeling Approaches

Exploring hybrid models that combine TCNs with other architectures such as RNNs or attention mechanisms could improve the capture of long-range dependencies and subtle temporal cues. Research in this area [1] suggests that such combinations can lead to more nuanced sequence modeling.

6.3.3 Real-Time Optimization

Enhancing the system for real-time performance is a key goal. Optimizing inference speed without sacrificing accuracy through model pruning or quantization techniques will be critical for deployment in live competitive environments.

6.3.4 User Feedback and Iterative Refinement

Incorporating structured feedback from judges and athletes will be essential in refining the system. Future iterations may include more granular feedback on detected infractions, offering not only binary outcomes but also detailed suggestions for corrective actions.

By addressing the current challenges and exploring new avenues for improvement, this project has the potential to redefine the standards of sports judging, making competitions fairer and more transparent.

CHAPTER 7

Conclusion

The AI Deadlift Judge project is a stride toward objective, automated evaluation in competitive powerlifting. By integrating advanced computer vision techniques with a custom-designed TCN for sequence modeling, the system effectively detects critical technical rule violations.

7.1 Summary of Contributions

7.1.1 Objective Judging Framework

The system replaces subjective manual assessments with an objective, data-driven process. By basing decisions on normalized key point data and temporal patterns, it offers consistent and repeatable evaluations, thus enhancing fairness and transparency in judging.

7.1.2 Integrated End-to-End Pipeline

An end-to-end solution was developed that encompasses video processing, pose extraction, sequence standardization, and temporal analysis, all orchestrated through a modular design. This pipeline, implemented in PyTorch and demonstrated via a user-friendly Streamlit interface, facilitates real-time feedback essential for live competitions.

7.1.3 Manually Curated Training Dataset

A unique aspect of this project is the manually created training dataset. This dataset was meticulously curated and annotated to capture a diverse range of deadlift attempts and associated infractions. The high-quality, manually verified labels have been critical in training the TCN to recognize subtle technical nuances reliably.

7.1.4 Robust Data Preprocessing and Augmentation

Standardizing videos to a fixed sequence and applying data augmentation techniques have significantly improved the model's robustness. These preprocessing strategies mitigate issues stemming from variable video quality and camera angles.

7.1.5 Comprehensive Evaluation

The system has been rigorously evaluated using both standard metrics — accuracy (80.00%), precision (74.18%), recall (92.31%), F1 score (82.15%) — and a custom judge score (83.33%). Both quantitative results and qualitative assessments show that the system is well-aligned with the expectations of objective deadlift judging.

7.1.6 Lessons Learned and Impact

The development and deployment of AI Deadlift Judge have underscored several key insights:

- Integrating modern deep learning with meticulous data curation leads to more reliable automated systems.
- Robust preprocessing and data augmentation are essential to manage real-world variability in video inputs.
- A modular, end-to-end pipeline not only simplifies system maintenance but also facilitates real-time applications.

In conclusion, the AI Deadlift Judge project sets a new benchmark in automated sports judging by combining modern computer vision and deep learning with a meticulously curated training dataset. The promising results achieved so far lay a solid foundation for future enhancements, with the potential to revolutionize the fairness and accuracy of judging competitive powerlifting.

Bibliography

[1] S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[2] Z. Cao, T. Simon, S. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[3] Y. Cui, M. Jia, T. Lin, Y. Song, and S. Belongie. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[5] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[6] C. Lea, M. D. Flynn, R. Vidal, and G. D. Hager. Temporal convolutional networks for action segmentation and detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[7] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[8] M. Lin, Q. Chen, and S. Yan. Network in network. In *International Conference on Learning Representations (ICLR)*, 2014.

[9] P. Miller and D. Thomas. Subjectivity in sports judging: Analysis of bias in artistic swimming. *Journal of Sports Sciences*, 22(11):895–903, 2004.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] D. M. W. Powers. Evaluation: From precision, recall and f-measure to roc, informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[14] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[16] Streamlit Inc. Streamlit – the fastest way to build data apps in python. `https://streamlit.io`, 2020.

[17] B. Xiao, H. Wu, and Y. Wei. Simple baselines for human pose estimation and tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

CHAPTER 8

Appendices

Project Repository

The complete source code and related materials for this project are available at:

`https://git.cs.bham.ac.uk/projects-2024-25/mxa1377`