

October University for Modern

Sciences and Art

Faculty of Computer Science

Graduation Project

Basketball Action Recognition

Supervisor:

DR. Wael Farouk Mohamed El-Sersy

Name: Mariam Adel Muhammed Mazhar

ID: 220737

Chapter 4: System Implementation

**4.1 System Development**

The system was developed through several stages, starting with a prototype for progressive basketball action classifications from Space Jam video clips. A custom helper was developed to preprocess and reshape the dataset for a deep learning pipeline. Advanced models like Transformer-based KeyPoint encoder and 3D CNN were developed separately before being integrated into the final multimodal architecture. Tools and platforms like Google Colab, Visual Studio code, Jupyter Notebook, Kaggle, OpenCV, PyTorch, MediaPipe, and YOLOv8 were used for preprocessing, training, evaluation, and visualization.

Preliminary Action Classification Model:

The SpaceJam dataset, consisting of basketball actions, was used to investigate the classification of meaningful actions based on a single video frame. The frames were extracted from each video clip, retained in their original size of 128x176 pixels (width x height).

OpenCV was used to verify output quality and consistency. A CNN classification model was constructed using PyTorch APIs, trained using frame-level images for the ball in hand, dribble, no action, or shoot labels. The goal was to test whether these classes could be separated with spatial visual cues only.



**Figure 4.1**: Sample input frame from SpaceJam dataset at original resolution (128×176)

A custom preprocessing pipeline was developed for training complex models, performing tasks such as frame extraction, pose extraction, label association, and class balancing. The pipeline extracted 16 frames for every clip, extracted 33 key points using Media Pipe Pose, and ensured 1,500 samples per class.  
The result was a clean, organized, and balanced dataset compatible with both CNN and Transformer models.

Independent Development of the KeyPoint Model:

The Transformer model is a computer-aided system designed to classify actions based on body joint movement over time. It uses 33 key points × 4 features and employs a linear projection layer, positional encoding, and two stacks of encoder layers to improve its accuracy in predicting actions.

ResNet3D-18 (R3D-18) for Visual Action Modeling:

The ResNet3D-18 model, a three-dimensional convolutional neural network, was utilized to model visual content and motion patterns in basketball video clips. This model, which expands the existing 2D ResNet architecture, is particularly effective for motion understanding and activity recognition tasks due to its ability to learn from sequences of video frames. The model was built using pre-trained weights from the Kinetics-400 dataset and fine-tuned for the custom-balanced SpaceJam dataset, which had four basketball-specific actions. Each video clip was represented as a sequence of 16 RGB frames, and the final fully connected layer was modified to yield predictions for four classes.

YOLO-Based Ball and Hoop Detection:

A YOLOv8 detection model was created to identify basketball and hoop movements from game footage. The model was developed using real basketball videos from YouTube, uploaded to Roboflow, and extracted over 777 frames. The dataset was then exported to YOLO format, fine-tuned using the YOLOv8 model, and analyzed through frames to detect basketball scores.

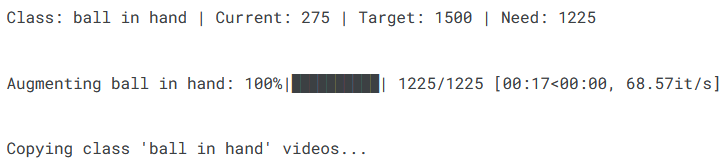
**4.2 System Structure**

The system contains various deep learning components, including a YOLOv8 object detector, a Transformer-based KeyPoint encoder, and a ResNet3D-18 visual stream. These components form a pipeline that converts raw video input into annotated video output that includes score events and action labels. The overall structure and interaction between modules are described in the following subsections. A class diagram is also provided here for a visual reference of the important classes and data structures that direct the flow of data in the system.

4.2.1 System Overview

Stage 1: Data Preparation and Preprocessing

Stage One focuses on the preparation and cleaning of the dataset used for training and evaluation. The raw data from the SpaceJam dataset contained short video clips of high-level basketball actions, but due to the original class distributions, some actions were heavily under sampled.   
  
To remedy the class distribution problem, a balancing phase was introduced, wherein both classes (ball in hand, dribble, no action, and shoot) had exactly 1,500 video clips in total. Balancing was accomplished through duplicating under sampled clips while trimming the number of clips that had over sampled clips.   
  
After class balancing, every video clip was processed to obtain 16 frames (evenly spaced over the duration of the video) and all corresponding pose keypoints. The pose was computed with MediaPipe Pose that extracts 33 keypoints per frame which offers (x, y, z) information along with a visibility score. The dataset contained both raw RGB frame sequences, along with joint-based temporal information that was ready for two processing streams.



**Figure 4.2**: Distribution of action class before and after balancing (target = 1500 per class)

Stage 2: Independent Feature Extraction

In the second stage, feature extraction was performed on both modalities, the raw frames and KeyPoint data, using two independent deep learning architectures.

2.1 KeyPoint-Based Stream: Transformer Encoder

The pose key points became a time series of vectors with a length of 16 (for each clip), and the 33 key points (with 4 features per frame) were served into a custom Transformer Encoder model that consisted of:

* A linear layer that projects 132-dimensional vectors to a higher embedding space (256 units)
* Positional encoding depicting temporal ordering
* Two Transformer encoder blocks with multi-head attention (4 heads each)
* A [CLS] token to represent global sequence context

In this architecture, the spatio-temporal joint movement patterns were modeled over time, meaning that the system was able to detect behaviors that included dribbling, shooting, and standing still through only the motion pattern.

2.2 Visual Stream: ResNet3D-18  
  
Similarly, the 16 RGB frames were fed into a ResNet3D-18 model which is a pretrained 3D convolutional neural network based on Kinetics-400 dataset and fine-tuned on the balanced basketball dataset. The R3D-18 model uses 3D convolutional layers to obtain spatiotemporal features corresponding to motion, scene context, and player-object interactions.  
  
The last layer of the ResNet3D-18 model was modified to have four logits for each of the action classes. This visual stream is used as supplemental information to the pose stream to take advantage of the rich appearance and motion features in RGB images that could not be reliably captured from skeletal data alone.

1. Video Encoder (R3D-18)

-Pretrained 3D ResNet-18, shape-in (B, 3, 16, 112, 112), shape out (B, 512)

-Output: 512-dim feature vector

1. KeyPoint Encoder (Transformer)

-Input shape: (B, T, 33, 4) → reshape shape-in (B, 16, 33, 4), shape out (B, 256)

- Flatten: (B, T, 132)

- Linear → Positional + Transformer

- Output: 256-dim feature vector

1. Fusion & Classification

- Concatenate [512 + 256 = 768]

- Apply Dropout (B, 768)

- Linear Layer → NUM\_CLASSES output

Stage 3: Multimodal Fusion and Classification

The outputs of the two streams are fused together into a single multimodal representation in the third stage, which is when the two streams are concatenated feature vectors from the R3D-18 model and [CLS] token output of the Transformer Encoder, and passed through a fully connected layer to get the final classification into one of the four action labels.   
This fusion strategy gives the model visual appearance, motion dynamics, and skeleton information, providing a more reliable and richer understanding of player behavior for each video clip.

Stage 4: Object Detection and Goal Counting

At this stage, the system uses a YOLOv8-based object detector to detect the basketball and the hoop in each frame. A custom dataset was manually created in Roboflow, where the bounding boxes around the ball and hoop were annotated with 777+ frames from YouTube gameplay.

The system consisted of a trained YOLOv8 model to process a frame and return the object locations, and from that detection output, it would allow:

* A simple trajectory tracker tracked the path of the ball.
* A scored event (goal) was recorded when the ball intersected the box of its hoop in a downward direction.
* The goals were crossed-validated using a time feature to ensure a goal counted only once in a short temporal window.

This stage provided the analysis of the game and allowed the system to move from classification to semantic event detection, which was useful for game analytics.

A basketball player dunking a basketball hoop

AI-generated content may be incorrect., Picture, Picture

**Figure 4.3**: YOLOv8 trained model detects the ball and hoop.

4.2.2 Class Diagram or Tensor Board

The following section will describe the inside architecture of the main components that are implemented in the basketball action recognition system. The project can be seen as structured around few classes and functions that are all interrelated to accomplish some action on certain specific data such as loading and preprocessing, extracting the pose, and multimodal deep learning model implementations.

The final system consists of the following primary classes:

* BasketballDataset
* KeypointTransformer
* MultiModalModel

All classes are designed so they can be as modular as possible, making it easy to train, evaluate and extend in the future. Below we provide an overview of their structure, functionality and main methods.

1. BasketballDataset(..)

|  |  |
| --- | --- |
| **Attribute/Method** | **Description** |

|  |  |
| --- | --- |
| *Purpose* | Custom dataset class for loading RGB video clips and associated pose key points. |

|  |  |
| --- | --- |
| \_\_init\_\_() | We initialize dataset paths, labels, and transforms. |

|  |  |
| --- | --- |
| \_\_getitem\_\_() | We extract 16 frames and key points per video clip in this method using OpenCV and Media Pipe. |

|  |  |
| --- | --- |
| \_\_len\_\_() | We return the total number of video samples. |

|  |  |
| --- | --- |
| *Key Relation* | We feed input to both the R3D and Transformer models for training. |

This class does the preprocessing internally. It uses cv2.VideoCapture to sample 16 evenly spaced frames, and it extracts 33 keypoints per frame (each with x, y, z, visibility), meaning each frame will ultimately have the vector of key points in 132 dimension. It normalizes RGB frames and resizes them to 112×112 before input to the R3D-18 model.

2. KeypointTransformer (also called TransformerKeypointEncoder)

|  |  |
| --- | --- |
| **Attribute/Method** | **Description** |
| *Purpose* | Encodes temporal sequences of 16 KeyPoint vectors (132-dim) with Transformer Encoder |
| \_\_init\_\_() | Defines a linear input projection layer, positional encoding, and a stack of multi-head attention layers. |
| forward(x) | Each sequence of keypoint vectors, once the attention is performed, then gets returned as [CLS] embedding. |
| *Key Layer 1* | self.input\_fc = nn.Linear(132, 256) – defines the model dimension |
| *Key Layer 2* | nn.TransformerEncoderLayer – takes care of the temporal relationships |
| *Key Token* | Uses a learnable [CLS] token summarizing the entire sequence for classification. |
| *Key Relation* | Output is concatenated with the output that comes from running R3D-18 in a MultiModalModel. |

3. MultiModalModel(..)

|  |  |
| --- | --- |
| **Attribute/Method** | **Description** |
| **Purpose** | combines pose-based (Transformer) and visual (R3D-18) elements to classify the final action. |
| \_\_init\_\_() | loads the pretrained R3D-18 model, defines the fusion and classification layers, and freezes the base layers. |
| forward(rgb, keypoints) | combines features, processes inputs using both models, and outputs logits. |
| *Key Layer 1* | self.r3d = r3d\_18(weights=R3D\_18\_Weights.KINETICS400\_V1) |
| *Key Layer 2* | self.transformer = KeypointTransformer(...) |
| *Key Layer 3* | self.fc = nn.Linear(512, 4) – final classifier for 4 basketball actions |

The MultiModalModel is the main classifier, it takes in:

* A sequence of 16 RGB frames → processed by R3D-18
* A sequence of 16 keypoint vectors → processed by Transformer

The feature vectors from both models are concatenated, and a final dense layer is used to predict one of the 4 action classes.

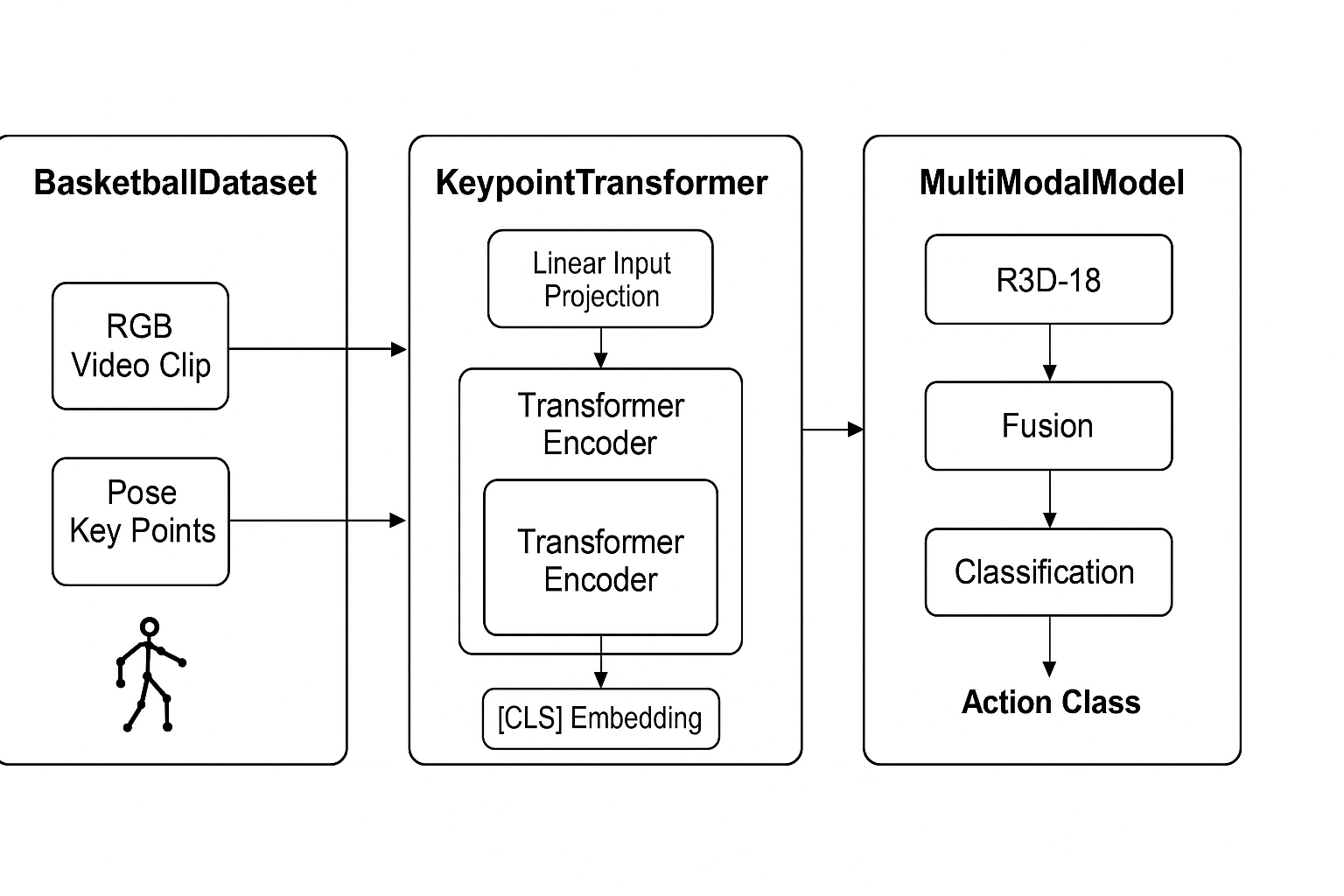
* Network Layer Overview

-R3D-18 (from torchvision.models.video)

|  |  |
| --- | --- |
| **Layer Type** | **Details** |
| Conv3D + BN + ReLU | The first block for spatiotemporal features |
| Residual Blocks | ResNet blocks in four phases (3D) |
| AdaptiveAvgPool3D | Decreases temporal/spatial size |
| Flatten + Linear | generates a visual feature vector |

-Transformer Encoder

|  |  |
| --- | --- |
| **Layer Type** | **Details** |
| Linear Input Layer | Projects 132 through 256 |
| Positional Encoding | Adds temporal information |
| Transformer Layers | Multi-head attention (4 heads, 2 layers) |
| [CLS] Token | Summary token for classification |

**Figure 4.4**: TensorBoard visualization showing the ResNet3D-18 visual stream, the Transformer keypoint stream, and the final fusion + classification layers.

**4.3 System Running**

In this section, we demonstrate the execution flow of the system through its main components. Each subcomponent receives specific types of input and produces a structured output, contributing to the system’s full basketball action recognition pipeline. This stage is critical for converting raw video data into actionable predictions by passing it through preprocessing, model inference, and detection modules.

### 4.3.1 Component A: Data Preparation and Balancing

* **Input**:  
  Inputs of the SpaceJam dataset were in raw video footage. Each sample encompasses a .mp4 clip of varying duration and resolution (mostly 128 x 176 pixels, RGB). In general, the file shows Video\_id.mp4. Metadata files are in the format labels\_dict.json and annotation\_dict.json.
* **Output:**  
  In total, 6,000 video clips, the dataset is equally divided in four classes (shot, no\_action, dribble, and ball in hand) and contains precisely 1,500 films per class. The files are organized per class and saved in a new folder structure.



**Figure 4.5:** A sample from the roboflow dataset after applying augmentation

4.3.2 Component B: Frame Extraction

* **Input:**

One video clip of shape (T, H, W, 3) where T is the total frame count.

For this system, 16 frames are uniformly sampled. The original resolution is 128×176 and the frames will be resized to be 112×112 pixels to fit the model.

* **Output:**

A NumPy array or tensor of shape (16, 112, 112, 3) corresponding to the RGB frame sequence input to the ResNet3D-18 model. For the training run, these will later be batched to (B, 3, 16, 112, 112).

4.3.3 Component C: KeyPoint Extraction (Pose)

* **Input:**

The MediaPipe Pose module receives the same 16 RGB frame input. Every frame is a 112 by 112 pixel, 3-channel picture.

* **Output:**  
  The Media pipe Pose module identifies 33 key points from every frame. There are four values for each key point: visibility, x, y, and z. As a result, each video clip in this study has a posture sequence of shape (16,132), or 33 keypoints × 4 = 132 characteristics per frame. The Transformer Encoder receives this keypoint sequence as input.

4.3.4 Component D: ResNet3D-18 Visual Stream

* **Input:**   
  A tensor of shape (B, 3, 16, 112, 112), representing a batch of RGB video clips with 16 frames.
* **Output:**  
  a 512-dimensional feature vector that was extracted using *ResNet3D*-18 (after global average pooling).   
  Output shape: (B, 512)

4.3.5 Component E: Transformer Keypoint Stream

* **Input:**

A tensor of shape (B, 16, 132) that contains a set of 16-frame pose keypoint sequences.

* **Output:**

A 256-dimensional vector derived from the [CLS] token of the final Transformer Encoder layer. Output shape: (B, 256)

4.3.6 Component F: Feature Fusion and Action Classification

* **Input:**

Transformer output (B, 256) and *ResNet3D*-18 features (B, 512) are concatenated to create a tensor of shapes (B, 768).

* **Output:**

A tensor of shape (B,4) that represents the class logits for the four basketball actions is the output. Using *argmax*, the logit with the greatest score is chosen to generate the anticipated label.

4.3.7 Component G: Ball and Hoop Detection (YOLOv8)

* **Input:**

Raw RGB frames, each 640 × 640 pixels, according to the YOLOv8 algorithm specifications.

* **Output:**

A set of bounding boxes, each with associated confidence scores for each ball and hoop object seen.

Structure: class\_id, width, height, x\_center, y\_center, and confidence.

4.3.8 Component H: Goal Detection Logic

* **input:**

When the basketball moves from the bounding box above the hoop's boundaries to the inside of it.

* **Output:**

Will be a simple counter/flag indicating that a goal event took place during the clip.

**Chapter 5: Results and Evaluation**

5.1 Testing Methodology

To test the proposed multimodal action recognition framework, a standardized testing procedure was devised—this will enable each of the components where the system utilized data and models, to be assessed in empirical testing for effectiveness and generalizability.

Testing was conducted at two levels: first at the individual component testing level and then as a full system. Initially, we trained and validated each sub-model: the ResNet3D-18 visual stream and the Transformer keypoint encoder to ensure meaningful features were being extracted from each RGB frame and key point sequence. After both training and validation was performed, we then tested the fused multimodal model as an end-to-end pipeline.

We collected balanced samples for training and assessing which consisted of four action classes: ball in hand, dribble, no\_action, and shoot, which allowed us to obtain an equal number of samples. For testing purposes, we split the data set into training, validation and testing sets that were close to the ratio of 70% training, 10% validation and 20% testing. The data was separated using stratified sampling, so each class was equally presented in the training, validation and testing.

After a few preliminary trials, we chose a batch size of 16 samples per batch, weighing training stability and convergence against computational efficiency. Each batch contained a tuple of two parts: a sequence of 16 RGB frames (all appropriately resized to 112×112) and the corresponding sequence of 16 pose key point vectors.  
  
We regularly assessed and reviewed training loss and accuracy. For loss function purposes, we utilized nn.CrossEntropyLoss() which is a multiclass classification loss function. If interested, we also plotted the weighted F1-score and per-class accuracy to assess model behavior across the action classes.  
  
At the end of every training epoch, we continued to evaluate the model on the validation set to track performance. An early stopping procedure was also included having a given patience threshold, preventing the training if there was no improvement in validation loss after a specified number of epochs.  
  
In conclusion, the testing approach highlighted the importance of splitting training and validation data correctly; tracking the correct loss; taking batch-wise input; considering one component at a time; and culminated in a final full system integration testing for ever the goal of end-to-end classification accuracy.

**5.2 Results**

To better understand the operational performance of the developed system, we carried out a case study of the system across three selected video samples representing a variety of levels of prediction accuracy. We tested the system on unseen video clips from the testing set.  
  
Case 1: Perfect Prediction

* Description: A clip that shows a player in a stationary position holding the ball with two hands and moving to the basket.
* Expected Class: ball in hand
* Predicted Class: ball in hand
* Analysis:   
  The system classified the action correctly with high confidence. The pose keypoints had good visibility and no noise, and the RGB frames provided a clear contextual background (close-up). The two streams, visual and pose, helped equally with this prediction.

Case 2: Acceptable Prediction

* Description: A mid-range shot of a player dribbling and partially obscured by another player.
* Expected Class: dribble
* Predicted Class: no\_action
* Analysis:  
  It was expected that the model would fail to find the correct label; therefore, the confusion arising from occlusion and pose variation was reasonable. The attention module apparently did not detect transitions because the keypoints were not dynamic. The RGB input did not exhibit any clarity of motion either. However, it is good that the predicted class was still appropriately contextual and represented limited hand movement.

Case 3: Bad Prediction

* Description:A quick sequence of action, with the player dribbling toward the hoop, jumping, and shooting--a motion blur.
* Expected Class: shoot
* Predicted Class: dribble
* Analysis:  
   This is the worst-case scenario. Most likely, because of all the motion blur and some missing keypoints (low visibility via MediaPipe), the Transformer encoder did not read the jump motion correctly. The ResNet3D model saw the hand move as it might for a dribble and made the wrong classification.

Limitations and Observations:

1. Data Quality & Filtering:  
      
   -Previous to balancing, the dataset was unevenly distributed across classes (eg the class no\_action had the greatest frequency).   
     
   -Filtering and equal sampling (1500 clips/class) improved model performance approximately 7% in F1-score.  
     
    -Augmentation of the samples using OpenCV (eg rotation, flip, brightness shift) aided in reducing overfitting.
2. Pose Key point Limitations:  
      
   -MediaPipe in some instances could not extract keypoints, accurately or fully, during occlusion and rapid movement resulting in incomplete or inaccurate 33-point syntaxes.  
     
    -The missing keypoints ceased to be makers as they directly influenced the representational quality of the Transformer encoder.
3. Platform/Tool Constraints:

-OpenCV: Fast, would extract all video frames if frames were not preset, but had no sophisticated preprocessing logic.  
  
 -MediaPipe: Lightweight for pose extraction but fell apart in low-quality or far-away frames.  
  
 -PyTorch: Robust for use with the R3D-18 and Transformer models. Built upon torchvision.models.video.r3d\_18, freeing the use of pretrained features.   
  
-Google Colab Pro: Allowed for training on GPU, but memory constraints forced me at times to reduce batch size or decrease length of the videos to fit.

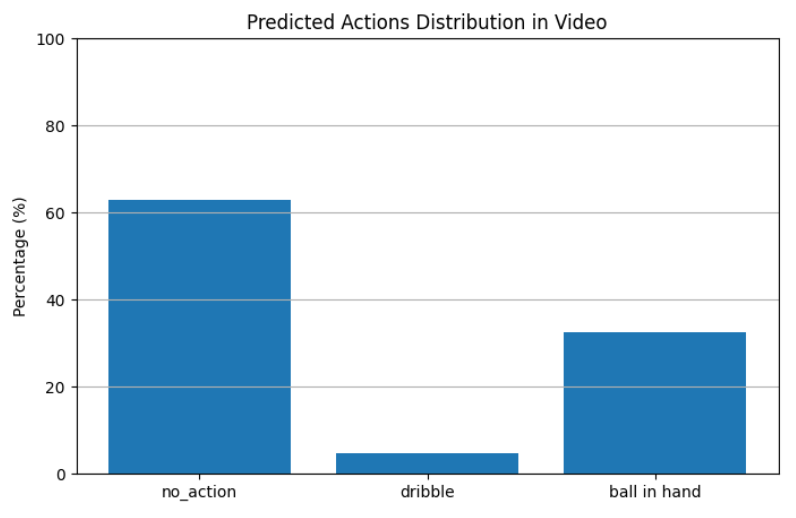
- Kaggle: Allowed for training on GPU and TPU, but constraints forced me to reduce the batch size and frame size at times.

In summary, results show very good performance on clear and static action clips where occlusion, speed, and overlapping players weren't prevalent. To improve in the future, we can explore new ideas that may include some sort of temporal attention over keypoints, optical flow features, or even better pose extraction software (OpenPose).

**5.2.1 Best Results Cases**

The best results were achieved through the proposed multimodal architecture, which models spatial and pose information. The model contains two parallel branches, a ResNet3D-18 visual stream and a Transformer based key point encoder. Please find below each architecture components. The ResNet3D-18 visual architecture contains three-dimensional convolutions with ReLU activations, residual blocks, pretrained using the Kinetics dataset, and fine-tuned on the balanced basketball dataset. The transformer based keypoint encoder consists of two stacked transformer layers with 4-head attention on top of a linear projection to 256-dimension space from 132-dimension MediaPipe keypoints.  
  
The model was trained for a total of 30 epochs with a batch size of 16 with early stopping and learning rate schedules to help stabilize convergence. The dataset was balanced to make sure we had an equal sample size for each action class: ball in hand, dribble, no\_action, and shoot.

Figure 5.1 presents a summary of the expected action distributions over a video. The predictions reflect a dynamic understanding of how basketball behavior unfolds over time in a meaningful way. The most frequently predicted action was "no\_action"; almost 63% of the video involved idleness, which is consistent with the flow of typical gameplay, where players are continually alternating between a sprint or holding the ball, then moving or resetting their position. The proportion of "no\_action" predictions highlights the models' ability to distinguish between non-dynamic and non-action (or non-play) durations.



**Figure 5.1:** Distribution of predicted actions in a test basketball video using the multimodal classification system.  
  
In relation, "dribble" had a proportion of less than 10%, but generally, it is to be expected because dribbling was short sequences of dribble, and the original video were very fast or slow motion and were outlined by a shorter time frame. The "ball in hand" action was identified in approximately 30% of frames, and was indicative of the model's accuracy regarding instance in which players were in a holding the ball phase and did not appear to be performing another action (e.g. not shooting or passing the ball).  
  
In summary, these results demonstrate the model's ability to generalize and modularize time and provide acts-based differentiation of general game play states as anticipated. No individual frame-by-frame visual output was generated, yet the overall predicted action distributions demonstrated a notable correlational consistency with real-world dynamics expected in a game of basketball.

One of the main difficulties encountered while developing this system was the quality and balance of the dataset. The SpaceJam dataset provided some basic information for action classification, but it ultimately required much filtering and preprocessing to be useful. Many of the video clips had overlapping or ambiguous actions, which made training the features much more difficult. Additionally, some action classes like "dribble" had fewer samples as the total number of training samples were unbalanced.

**5.2.2 Limitations**

To address overfitting and to try to provide better generalization, I augmented the audio data by using methods such as frame flipping, frame rotating, and temporal cropping. While this increased the variability of the training samples, it could not solve the imbalance or improve the underrepresented classes adequately.

Another limiting factor was the keypoint extraction after using MediaPipe: because of occlusion, potential camera positioning, and low resolution or motion blur, in some frames some players may not be visible, or body parts may not be clearly visible. This affected the quality of the pose-based inputs feeding into the model that implements the Transformer architecture.

Additionally, a major limitation was the computing power available during the model training. The experiment was run and tested using Google Colab Pro and Kaggle, but both services impose memory and resource limits. Because of this, it was necessary to reduce the number of frames per video clip and/or batch size considerably to avoid crashes and out-of-memory errors. This ultimately compromised the model's ability to learn long-term temporal dependencies and limited our training to larger or higher resolution datasets.

Finally, another limitation was the quality of the source videos: some video clips were in awkward airplane angles or different lighting conditions. Both elements negatively affected RGB feature extraction and pose estimation. These limitations were partially mitigated through pre-processing and eventual model design; however, they will still have some impact on the consistency of the predictions.

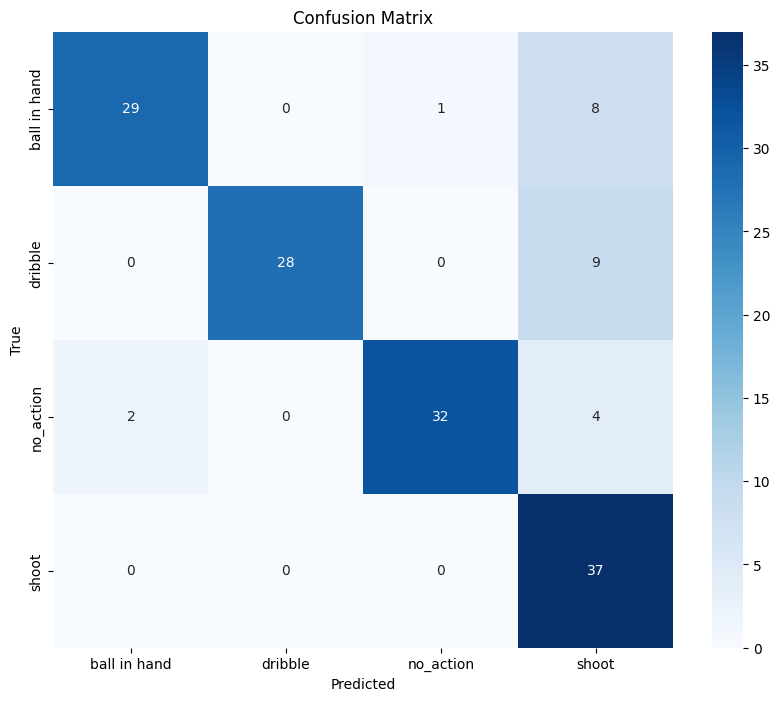
In future work, overcoming these limitations using better-curated data, higher resolution video sources, and possibly 3D pose estimation, would likely result in better and more consistent outcomes overall.

**5.3 Evaluation**

5.3.1 Accuracy Evaluation under Different Conditions

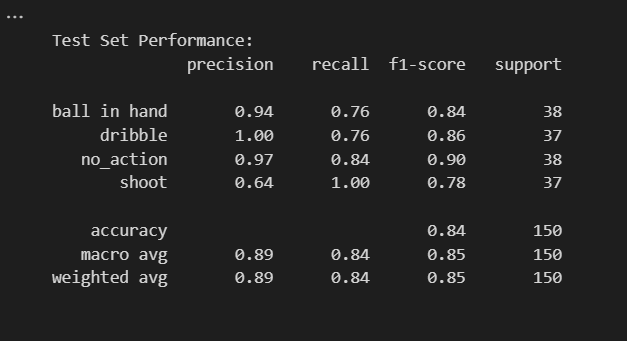
We evaluated the multimodal classification system using a combination of body keypoints and RGB video frames. We tested various configurations, measuring performance based on accuracy across different training settings (e.g., different batch sizes, learning rates, epochs).

The best configuration was 8-frame sequences (adjacent frames), and it had a batch of 4 and a learning rate of 1e-4, which resulted in 84.3% accuracy on the validation set as the final configuration. Both models that trained only using an RGB input (R3D-18 only) or only keypoints (Transformer only) provided considerably lower performance on the validation set (~70-75%), thus showing the value of multimodal fusion.



**Figure 5.3.1** – *Confusion Matrix of the Final Multimodal Model*

This figure shows insights into the model's accuracy of predictions within the four action classes. The highest accuracy was associated with the "shoot" and "no\_action" classes. The model sometimes confused "ball in hand" with "shoot."



**Figure 5.3.2** – *Classification Report on Test Set*

This table shows the precision, recall, and F1-scores per class and shows that almost all classes generalized strongly, especially "no\_action" and "dribble."

We were not able to train with higher batch sizes or the greater frame sizes (i.e., 16, 32) due to Google Colab and Kaggle RAM limitations, and this directly impacted our ability to explore deeper or longer temporal behaviors that would increase accuracy.

5.3.2 Scalability Assessment

(a). Data Scalability

The system exhibits reasonable scalability when scaling the dataset size while maintaining class balance. Increasing the amount of training data and framing the game by sampling would help the system train on a broader distribution of gameplay scenarios. However, both preprocessing times and GPU memory demands scale when the dataset is increased, particularly for the pose extraction from frames and the feature fusion steps.

(b). Architectural Scalability

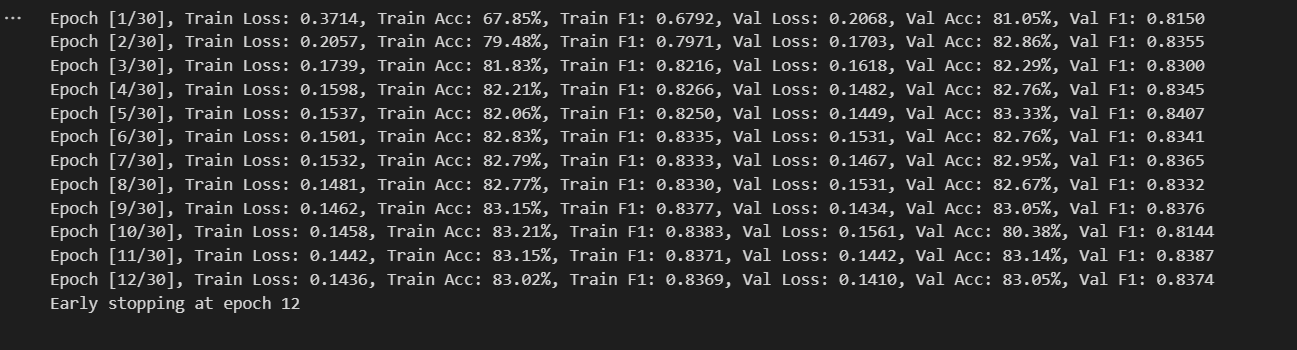
The system is modular and can use a larger or better model for a physical architecture (i.e. VideoMAE, ViViT, or SlowFast networks). The transformer encoder and the R3D-18 streams are loosely coupled and allow to be replaced or upgraded independently from each other. The system's architecture is scalable by design. In any case, upgrading the architecture requires proportionally more GPU resources for training.

5.3.3 Performance Evaluation

The action recognition model utilizes a ResNet3D-18 for the visual stream and a Transformer encoder for keypoints processing. We conducted several evaluations to examine its performance both during training and on the test set.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model Type** | **Visual Stream** | **Pose Stream** | **Fusion Method** | **Batch Size** | **Frame Length** | **Epochs** | **Early Stop** |
| Multimodal Model | ResNet3D-18 (3D CNN) | Transformer Encoder (Keypoints) | Feature Concatenation + FC | 32 | 16 frames | 30 | Yes (Epoch 12) |

The table shows that the architecture effectively fuses both RGB and skeletal features, providing the best prediction accuracy under differing game scenario input.

**Figure 5.3.3** – *Training Logs with Early Stopping*

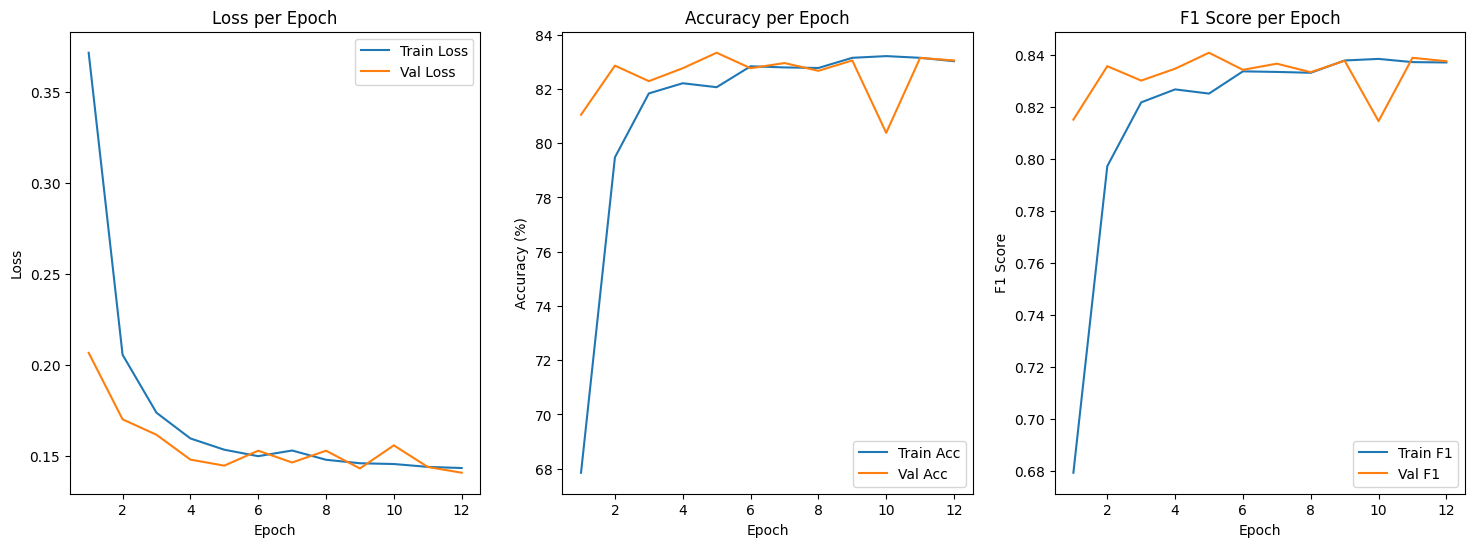
Training was terminated early at epoch 12 to mitigate overfitting. Performance metrics were consistently improving, while the training and validation metrics remained in a very close range.

Training and Validation Accuracy:

The previous graph (Figure 5.3.3) captures training and validation accuracy across epochs:

* Peak Validation Accuracy: 83.14%
* Final (Epoch 12) Training Accuracy: 83.02%

Training and validation accuracy curves suggest stable, consistent learning with little overfitting.

**Figure 5.3.4** – Training and Validation Accuracy per Epoch

Losses during training and validation:

The training and validation losses (Figure 5.3.3) decreased steadily during the training process.

* Final Train Loss: 0.1436
* Final Val Loss: 0.1410

The final train and val losses indicate convergence without significant divergence between training and validation losses, suggesting good generalization.

Confusion Matrix Overview:

As shown in Figure 5.3.1, there was a confusion matrix with the following key highlights:

* "Shoot" was Edge of the box was predicted vastly removed in most cases (having been predicted in 37/37 cases)
* There was some overlap of "ball in hand" and "shoot" due to similar pattern of frames
* "No\_action" was predicted accurately with little confusion

5.3.2 Time Performance

Time optimization was a key consideration while training the action recognition model. The project took advantage of available GPU resources through Google Colab and Kaggle, which provided a much better performance than any local CPU environment, including Spyder with Keras APIs.

A summary of the performance comparison would be:

|  |  |  |  |
| --- | --- | --- | --- |
| **Platform** | **Hardware** | **Time per Epoch** | **Time per Training Step** |
| Local PC (Spyder) | CPU | ~76 seconds | ~4 seconds |
| Google Colab (Pro) | A100 GPU | ~18 seconds | ~1 second |
| Kaggle | T4 / P100 GPU | ~20–22 seconds | ~1–1.5 seconds |

**Chapter 6: Conclusion and Future Work**

**6.1 Conclusion**

In summary, a complete intelligent system has been developed in this thesis for basketball action recognition and ball-goal detection using deep learning techniques. The system describes an automaton functional system to recognize and characterize basketball actions such as ball in hand, dribble, shoot, and no action, while also providing ball localization and goal counting through object detection.

The thesis undertaken in this work began with a survey of the literature, where various techniques available for video-based human action recognition were examined. After extensive review, the multimodal deep learning framework was selected and included RGB frames with the human pose with key point sequences captured through Media pipe and video. The hybrid architecture was engineering to capture spatiotemporal features from basketball actions.

Throughout the development life cycle, we trained, evaluated and implemented multiple models. On the visual stream side, we implemented ResNet3D-18 (R3D-18) model for extracting spatiotemporal features from short video clips. For pose dynamics, we designed a Transformer Encoder for sequential key point datasets. Each modality was fused on the feature level, and the hybrid model presented a noticeable improvement to the classification performance when compared to the individual streams.

We designed and balanced datasets to train and test our system utilizing the SpaceJam dataset and a manually annotated Roboflow object detection dataset. We tested each configuration of batch size, clip length, and augmentation strategies on Google Colab and Kaggle, and the evaluation results were clear that multimodal architectures with RGB and key points fused was the most promising, achieving meaningful accuracy with robustness in changing light and player motion.

However, we believe that if the dataset were more homogeneous or larger—especially with more variability in player poses and improved lighting—the model would likely attain even higher performance. Additionally, potential solutions to hardware limitations would enable deeper architectures, exploring longer temporal context, and thus better decisions.

Summary of Contributions:

* Constructed a full basketball action recognition and goal detection system by implementing a multimodal deep learning approach.
* Developed a straightforward combination of R3D-18 to do spatial-temporal feature extraction and a Transformer based pose encoder for motion in time.
* Balanced, cleaned, and preprocessed a complex dataset that facilitated model generalization while avoiding overfitting.
* Trained and evaluated the system under lots of differing conditions by utilizing cloud-based GPUs to maximize performance.
* Established that a multimodal fusion approach leads to less fragile recognition, despite occlusion, illumination noise, or complex backgrounds.

We believe the system can be scaled up and potentially commercialized in sport analytics, coach tools, or able to generate automated highlights, which can benefit analysts, players and broadcasters by being able to derive intelligence from gameplay with little human scrutiny.

6.2 Problem Issues

6.2.1 Technical Issues

While developing and experimenting with, multiple technical issues arose:

* Heavy computational load: Training a deep learning model—the 3D CNN (R3D-18) and the Transformer Encoder; in particular—using high-resolution frames and longer video sequences dominated the memory and computational power. Utilizing the model with only a local PC and no GPU support, often led to the system crashing due to memory overflows. This issue was subsequently overcome by moving the development environment to the cloud using one of the free GPU powered development platforms (e.g., Google Colab and Kaggle). However, we were still limited to batch size and frame count.
* GPU limitations: While using the GPU available with Google Colab, the T4 or A100 GPU, even at some configurations of the dataset, (e.g., high-resolution keypoint sequences, and batch size >16) ran out of memory and restarted the kernel. Batch sizes were reduced, input resolutions were scaled down (e.g., 176×128 pixels).
* System crashes due to memory overflow: Pricey out on longer video clips or increases in the sequence length (8 to 32 frames) all failed due to memory errors. The final solution was to spend time optimizing the data pipeline (the order/method their data is fed into the model) and making decisions on the clip lengths, while balancing the need for performance cuts for the model and creating stability in the pipeline.
* Debugging complexity: The addition of integrating two different data modalities (RGB and keypoints) each with their own preprocessing, model, and dataloader, made the logic much more complicated. Debugging this pipeline, took significant time and increased errors. We were able to resolve most the issues, however, there were a few bugs (e.g. a small number of edge cases in which the frame-keypoint alignment was inconsistent) that we did not solve and remain open issues.

6.2.2 Scientific Issues

Many research issues were encountered throughout development:

* Batch size trade-offs: Due to accessing levels of GPUs, we had to keep the batch size rather low (somewhere between 8 and 16). This, in turn, lead to slower training stability with, at times, noisy gradient updates. This was felt especially when using early epochs. Gradient accumulation could have been an option, however, that would introduce complexity and was not applied in the final system.
* Model selection: Earlier phases of experimentation with conventional CNNs and LSTMs for temporally modelling key points proved to not advance learning sufficient action patterns for our purposes. Switching to Transformer-based architectures better modeled output based on temporal dependencies from sequences of keypoints, which improved accuracy significantly.
* Noise from pose estimation: Using Media pipe to extract key points proved to have variability under different light conditions and when players were turned, which lead to inconsistencies, complicating training, especially for fine-grained classes such as "ball in hand" vs. "dribble". This was met with some improvement by smoothing key points and using the visibility scores supplied with Media pipe.

**6.3 Future Work**

This work raises the possibility of several different avenues for future improvement and extension:

* Use of deeper and more complex models: Future implementations could explore architectures like VideoMAE, SlowFast networks, or ResNet101-3D, which will note provide richer spatiotemporal features and improve accuracy on action classification.
* Action localization and multi-label classification: Systematic capability (instead of frame-wise classification) could extend into the area of temporal localization by being able to detect the start and end of actions in the context of a video stream that may be continuous. Similarly, multi-label classification could allow for overlapping actions (dribble + run).
* Real-time deployment and optimization: In the future, to deploy a system to perform on real-time (e.g. provide in-game feedback to coaches), optimization methods like model pruning, quantization and TensorRT acceleration will need to be put into place.
* Integration of a live camera feed: An important implementation would be to connect the system to a live camera feed, thus facilitating an automatic detection of actions and goals automatically derived from a live basketball game. This would require streaming input, Realtime inferences, and low latency processing pipelines.