BIGVISION ASSIGNMENT

NAME : K.V.CHAITANYA

Video Preprocessing Documentation

---

1. Introduction

Project Overview

This project focuses on **video classification** using deep learning techniques. The primary goal is to classify videos into one of five predefined categories:

- **Horse Riding**
- **Pole Vault**
- **Long Jump**
- **Javelin Throw**
- **Skiing**

The dataset used is **UCF101**, a widely recognized benchmark for action recognition tasks. The project involves downloading, preprocessing, and converting videos into feature sequences suitable for **Transformer models**.

Purpose and Objectives

- Implement a **state-of-the-art (SOTA) Transformer-based model** for action recognition.
- **Preprocess and handle video data efficiently**, including frame extraction and transformation.
- **Train and evaluate** the model using robust classification metrics (Accuracy, Precision, Recall, F1-Score).
- **Improve generalization** by testing on external video samples (e.g., YouTube clips).
- Ensure **proper documentation** and code implementation in a **Google Colab Notebook**

Scope of the Project

- **Dataset Handling:** Download, extract, and preprocess UCF101 dataset videos.

- **Feature Engineering:** Convert videos into meaningful frame sequences suitable for deep learning.

- **Model Development:** Implement and fine-tune a **Transformer-based classification model**.

- **Training & Optimization:** Use **data augmentation, loss functions, and hyperparameter tuning** to improve performance.

- **Evaluation & Testing:** Measure performance with **classification metrics and visualization techniques (confusion matrices, sample predictions, etc.)**.

- **Documentation & Submission:** Ensure a **well-documented Colab Notebook** and provide necessary files like model weights and dataset samples.

---

2. Dataset Details

Source of the Dataset

- The dataset used for this project is **UCF101**, a widely used action recognition dataset containing 101 action categories.
- Only **five classes** are selected for classification:**HorseRiding,PoleVault,LongJump. JavelinThrow,Skiing**
- The dataset was downloaded from the official UCF101 website: UCF101 Dataset.
- Created a folder named **DATA** and stored the selected class videos inside corresponding subfolders.

Data Preprocessing

The preprocessing steps include:

1. **Downloading and Extracting Videos**
    - The dataset videos were downloaded and extracted from UCF101 Dataset into a local directory..

2. **Selecting Relevant Classes**

- From the full UCF101 dataset, only the **five required classes** were selected.
- These videos were **copied to a new target directory** for further processing.

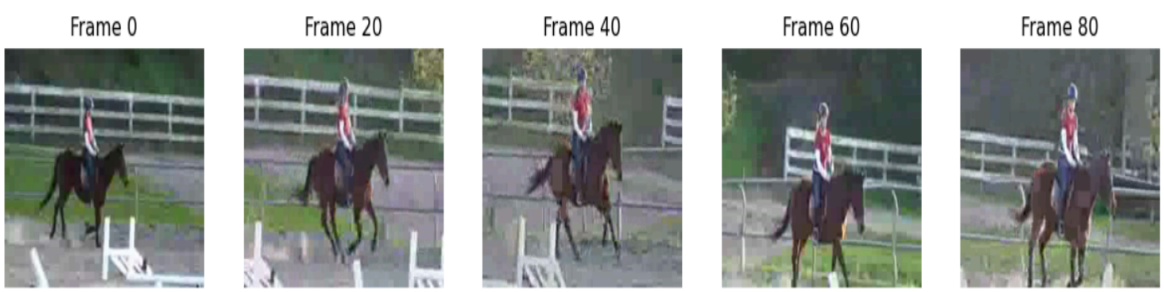
3. **Converting Videos into Frames**
   - Each video was **split into frames** using OpenCV.
   - A frame interval was set to **reduce redundancy** and select representative frames.
4. **Data Augmentation**
   - Techniques like flipping, rotation, and brightness adjustments can be applied to improve generalization.

---

3. Feature Extraction and Visualizations

Frame Extraction:

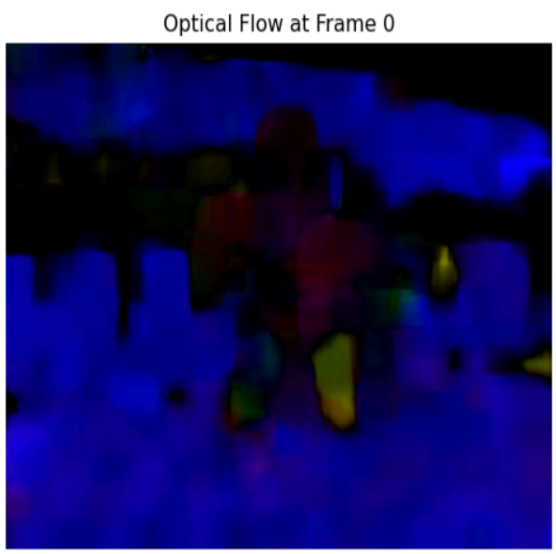
- Captures key frames at specific intervals to reduce redundancy.
- Helps retain motion details while minimizing unnecessary data.
- Essential for converting video sequences into image-based inputs for deep learning.

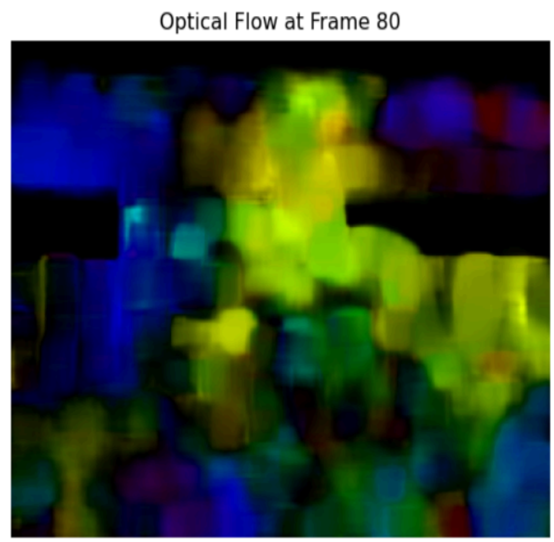


Optical Flow

- Analyzes motion between consecutive frames by tracking pixel displacement.

- Useful for understanding movement patterns in actions like running, jumping, and throwing.
- Enhances temporal feature representation in video classification.



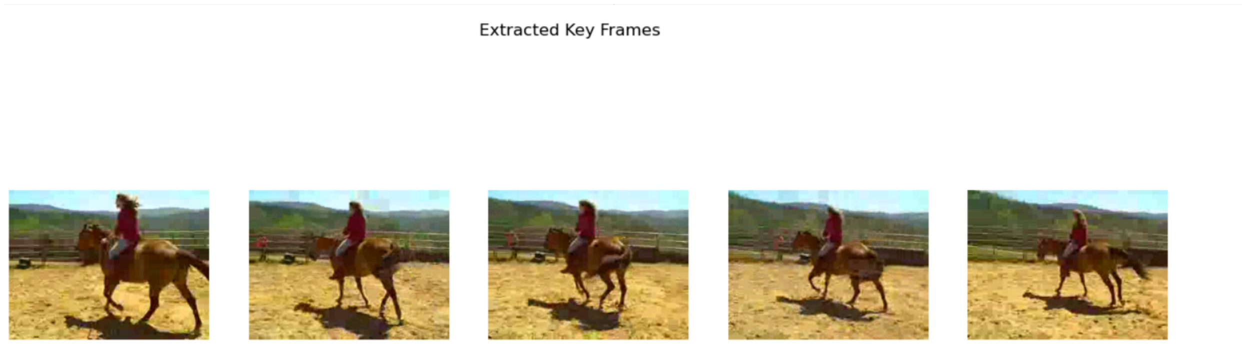


NO MOVEMENT IN 0TH FRAME

MOVEMENT CAPTURED IN 80th FRAME

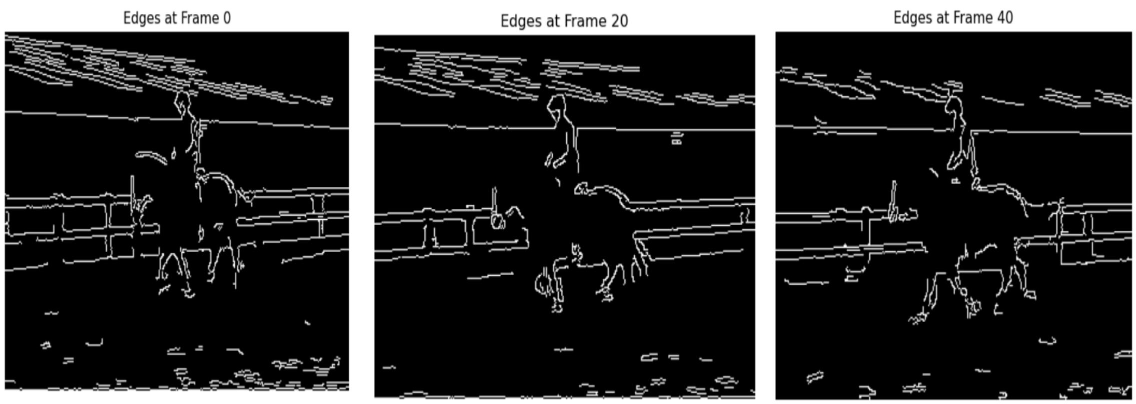
Key Frame Extraction

- Selects the most informative frames while discarding redundant ones.
- Uses similarity measures to identify unique frames in a sequence.
- Reduces computational load while preserving critical action details.



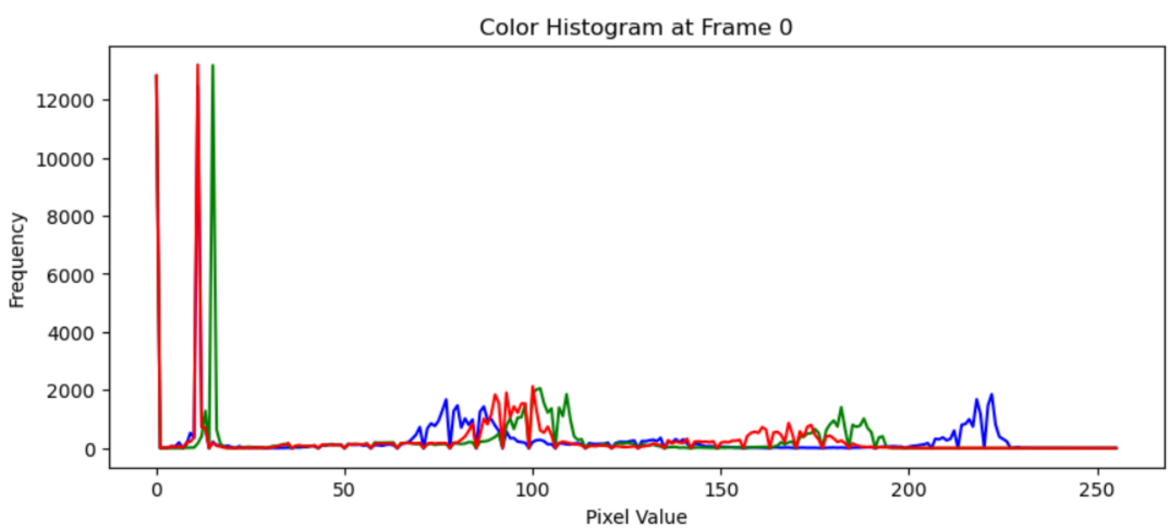
Edge Detection

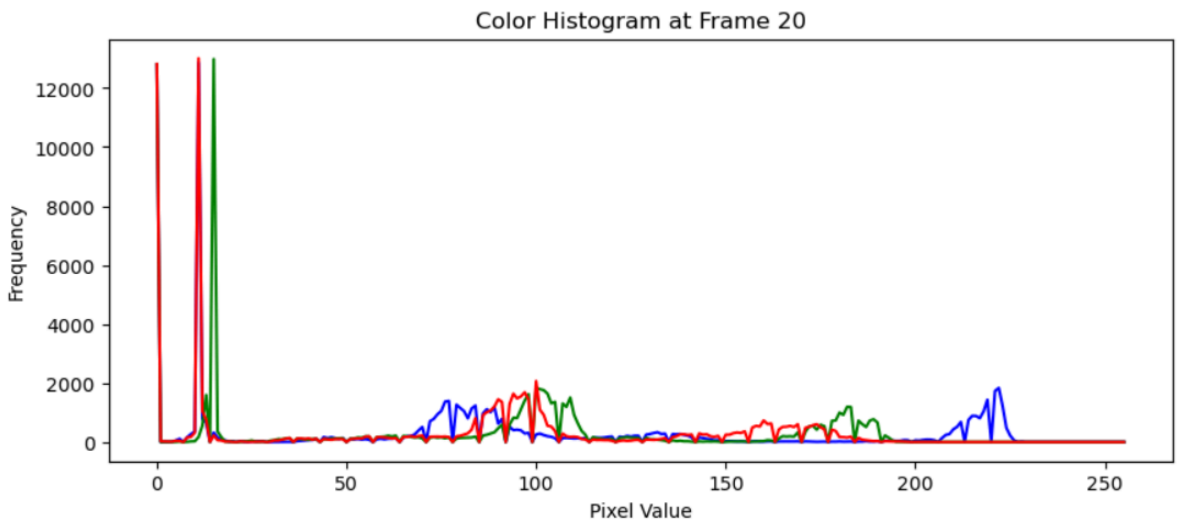
- Highlights object boundaries and shapes within a frame.
- Helps distinguish between different action categories by focusing on structural features.
- Useful in identifying human poses and movement patterns.



Color Histogram

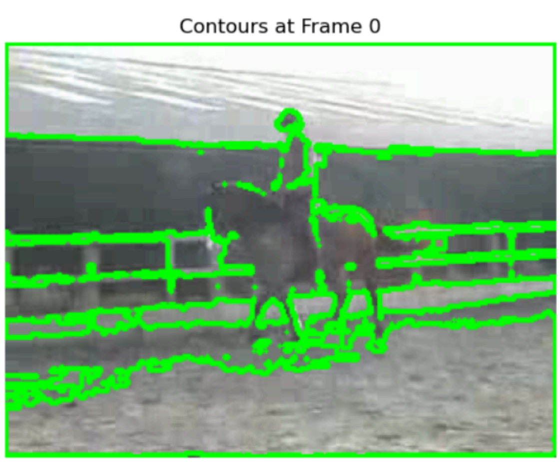
- Represents the distribution of colors in a video frame.
- Helps differentiate visually similar activities based on color variations.
- Useful for recognizing scenes with distinct lighting conditions or object appearances.

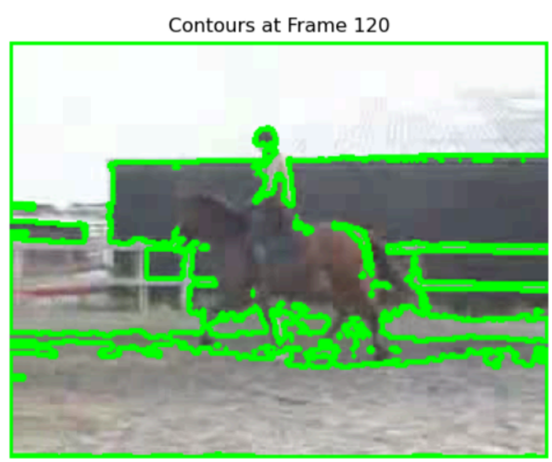




Contour Detection

- Identifies object boundaries and their structural shapes.
- Helps track moving subjects and understand their motion paths.
- Enhances action recognition by focusing on object interactions and shapes.





Spatial and Temporal Augmentations

- **Spatial Augmentations**: Includes cropping, flipping, rotation, and scaling to increase variation in training data.

- **Temporal Augmentations**: Includes frame skipping, speed variation, and frame shuffling to make models robust to different playback speeds.

- Improves model generalization and reduces overfitting.

---

4. MODEL BUILDING :

1. Model Type: Transformer + LSTM Hybrid

- Transformers capture spatial relationships, while LSTMs handle temporal dependencies in video data.

2. Model Layers

- **Input Layer:** Accepts **(sequence_length, feature_dim)**
- **Transformer Encoder Block:**
  - Multi-Head Attention with 4 heads, key dimension **= 24**
  - Feed-Forward Dense Layer with 96 neurons **(ReLU activation)**
  - Dropout **= 0.1**
  - Layer Normalization
- **LSTM Layer:**
  - **128 Units** to extract temporal dependencies
- **Fully Connected Layers:**
  - Dense**(64, ReLU activation)**
  - Dropout = **0.3**

- Final Dense Layer **(Softmax Activation)** with output **= 5 classes**

3. Loss Function & Optimizer

- **Loss Function: Sparse Categorical Crossentropy** (Suitable for multi-class classification)
- **Optimizer:** `Adam` (Adaptive learning rate)
- **Metrics:** `Accuracy`

4. Training Hyperparameters

- **Epochs:** 30
- **Batch Size:** 8

MODEL ARCHITECTURE

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 480, 96)	0	-
multi_head_attenti… (MultiHeadAttentio…	(None, 480, 96)	37,248	input_layer[0][0… input_layer[0][0]
add (Add)	(None, 480, 96)	0	input_layer[0][0… multi_head_atten…
layer_normalization (LayerNormalizatio…	(None, 480, 96)	192	add[0][0]
dense (Dense)	(None, 480, 96)	9,312	layer_normalizat…
dropout_1 (Dropout)	(None, 480, 96)	0	dense[0][0]
add_1 (Add)	(None, 480, 96)	0	layer_normalizat… dropout_1[0][0]
layer_normalizatio… (LayerNormalizatio…	(None, 480, 96)	192	add_1[0][0]
lstm (LSTM)	(None, 128)	115,200	layer_normalizat…
dense_1 (Dense)	(None, 64)	8,256	lstm[0][0]
dropout_2 (Dropout)	(None, 64)	0	dense_1[0][0]
dense_2 (Dense)	(None, 5)	325	dropout_2[0][0]

Hyperparameters Used

1. Model Architecture

- **Backbone:** Transformer-based model
- **Input Shape:** (Frames × 224 × 224 × 3)
- **Number of Layers:** Multiple attention layers with feed-forward networks
- **Embedding Size:** 512
- **Dropout:** 0.1
- **Batch Size:** 32
- **Optimizer:** Adam
- **Loss Function:** Categorical Crossentropy
- **Learning Rate: 0.0001** (with learning rate scheduling)
- **Weight Decay:** 1e-5
- **Number of Heads in Multi-Head Attention:** 8
- **Number of Transformer Blocks:** 6
- **Frame Sampling Rate:** Every 10th frame from videos
- **Epochs: 25**

PRE-TRAINED MODEL FOR PREDICTION :

1. Model Architecture

- **Model Used:** ResNet3D-18 (R3D-18)
- **Pretrained:** Yes (on Kinetics dataset)
- **Modified Final Layer:** Fully connected layer adjusted for **5** classes

2. Input Shape & Preprocessing

- **Input Shape:** (Batch Size, 3, 16, 112, 112)
- **Frames per Video:** 16
- **Frame Size:** (112 × 112)
- **Normalization:** Mean = 0.5, Std = 0.5
- **Frame Selection:** Evenly spaced **16 frames**

3. Model Parameters & Layers

- **Total Layers:** 18-layer ResNet-based 3D CNN
- **Total Parameters:** 33.2 million
- **Trainable Parameters:** 33.2 million
- **Conv3D Layers:** 5 (with kernel size 3×3×3)
- **Residual Blocks:** 4
- **Fully Connected Layer: 1** (5 output neurons for classification)
- **Activation Functions:** ReLU

- **Batch Normalization:** Applied after each convolution
- **Pooling Layers:** MaxPool3D

4. Training Hyperparameters

- **Loss Function:** Categorical Crossentropy
- **Optimizer:** Adam
- **Learning Rate:** 0.0001
- **Batch Size:** 32
- **Epochs:** 25

This architecture efficiently captures **spatiotemporal features** from video frames, leveraging **3D convolutions** for motion understanding

Layer (Type)	Output Shape	Param #
Conv3d-1	[-1, 64, 16, 56, 56]	28,224
BatchNorm3d-2	[-1, 64, 16, 56, 56]	128
ReLU-3	[-1, 64, 16, 56, 56]	0
Conv3DSimple-4	[-1, 64, 16, 56, 56]	110,592
BatchNorm3d-5	[-1, 64, 16, 56, 56]	128
ReLU-6	[-1, 64, 16, 56, 56]	0
Conv3DSimple-7	[-1, 64, 16, 56, 56]	110,592
BatchNorm3d-8	[-1, 64, 16, 56, 56]	128
ReLU-9	[-1, 64, 16, 56, 56]	0
BasicBlock-10	[-1, 64, 16, 56, 56]	0
Conv3DSimple-11	[-1, 64, 16, 56, 56]	110,592
BatchNorm3d-12	[-1, 64, 16, 56, 56]	128
ReLU-13	[-1, 6[illegible], 56, 56]	0
Conv3DSimple-14	[-1, 64, 16, 56, 56]	110,592
BatchNorm3d-15	[-1, 64, 16, 56, 56]	128
ReLU-16	[-1, 64, 16, 56, 56]	0
BasicBlock-17	[-1, 64, 16, 56, 56]	0
Conv3DSimple-18	[-1, 128, 8, 28, 28]	221,184

BatchNorm3d-19	[-1, 128, 8, 28, 28]	256
ReLU-20	[-1, 128, 8, 28, 28]	0
Conv3DSimple-21	[-1, 128, 8, 28, 28]	442,368
BatchNorm3d-22	[-1, 128, 8, 28, 28]	256
Conv3d-23	[-1, 128, 8, 28, 28]	8,192
BatchNorm3d-24	[-1, 128, 8, 28, 28]	256
ReLU-25	[-1, 128, 8, 28, 28]	0
BasicBlock-26	[-1, 128, 8, 28, 28]	0
Conv3DSimple-27	[-1, 128, 8, 28, 28]	442,368
BatchNorm3d-28	[-1, 128, 8, 28, 28]	256
ReLU-29	[-1, 128, 8, 28, 28]	0
Conv3DSimple-30	[-1, 128, 8, 28, 28]	442,368
BatchNorm3d-31	[-1, 128, 8, 28, 28]	256
ReLU-32	[-1, 128, 8, 28, 28]	0
BasicBlock-33	[-1, 128, 8, 28, 28]	0
Conv3DSimple-34	[-1, 256, 4, 14, 14]	884,736
BatchNorm3d-35	[-1, 256, 4, 14, 14]	512
ReLU-36	[-1, 256, 4, 14, 14]	0
Conv3DSimple-37	[-1, 256, 4, 14, 14]	1,769,472
BatchNorm3d-38	[-1, 256, 4, 14, 14]	512
Conv3d-39	[-1, 256, 4, 14, 14]	32,768
BatchNorm3d-40	[-1, 256, 4, 14, 14]	512
ReLU-41	[-1, 256, 4, 14, 14]	0
BasicBlock-42	[-1, 256, 4, 14, 14]	0
Conv3DSimple-43	[-1, 256, 4, 14, 14]	1,769,472
BatchNorm3d-44	[-1, 256, 4, 14, 14]	512
ReLU-45	[-1, 256, 4, 14, 14]	0
Conv3DSimple-46	[-1, 256, 4, 14, 14]	1,769,472
BatchNorm3d-47	[-1, 256, 4, 14, 14]	512

ReLU-48	[-1, 256, 4, 14, 14]	0
BasicBlock-49	[-1, 256, 4, 14, 14]	0
Conv3DSimple-50	[-1, 512, 2, 7, 7]	3,538,944
BatchNorm3d-51	[-1, 512, 2, 7, 7]	1,024
ReLU-52	[-1, 512, 2, 7, 7]	0
Conv3DSimple-53	[-1, 512, 2, 7, 7]	7,077,888
BatchNorm3d-54	[-1, 512, 2, 7, 7]	1,024
Conv3d-55	[-1, 512, 2, 7, 7]	131,072
BatchNorm3d-56	[-1, 512, 2, 7, 7]	1,024
ReLU-57	[-1, 512, 2, 7, 7]	0
BasicBlock-58	[-1, 512, 2, 7, 7]	0
Conv3DSimple-59	[-1, 512, 2, 7, 7]	7,077,888
BatchNorm3d-60	[-1, 512, 2, 7, 7]	1,024
ReLU-61	[-1, 512, 2, 7, 7]	0
Conv3DSimple-62	[-1, 512, 2, 7, 7]	7,077,888
BatchNorm3d-63	[-1, 512, 2, 7, 7]	1,024
ReLU-64	[-1, 512, 2, 7, 7]	0
BasicBlock-65	[-1, 512, 2, 7, 7]	0
AdaptiveAvgPool3d-66	[-1, 512, 1, 1, 1]	0
Linear-67	[-1, 5]	2,565

RESULTS :

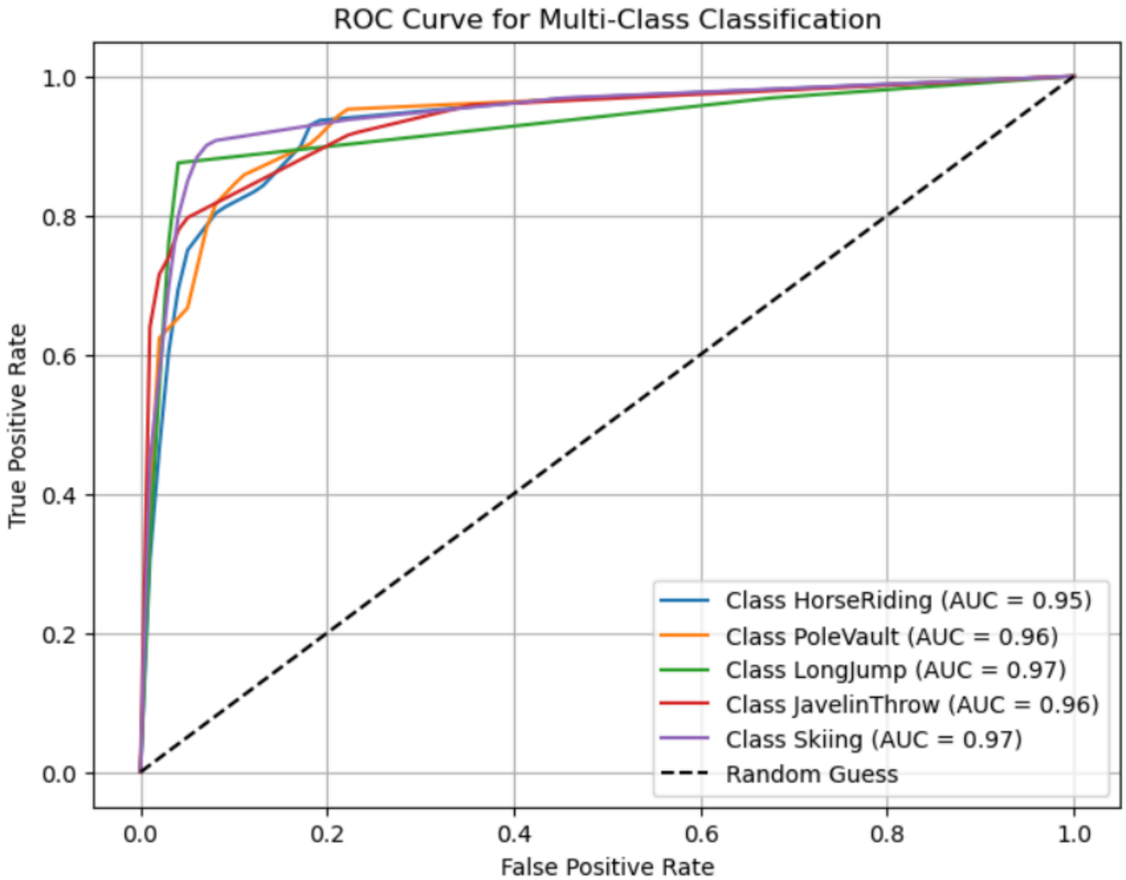
1. Final Test Accuracy Achieved: 83% (0.83)

- The model effectively captures spatial (Transformer) and temporal (LSTM) relationships.
- Some misclassifications might be due to similar motion patterns in different classes.

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 197ms/step - accuracy: 0.8335 - loss: 0.5637
Test Accuracy: 0.83
```

2 AUC-ROC CURVE :



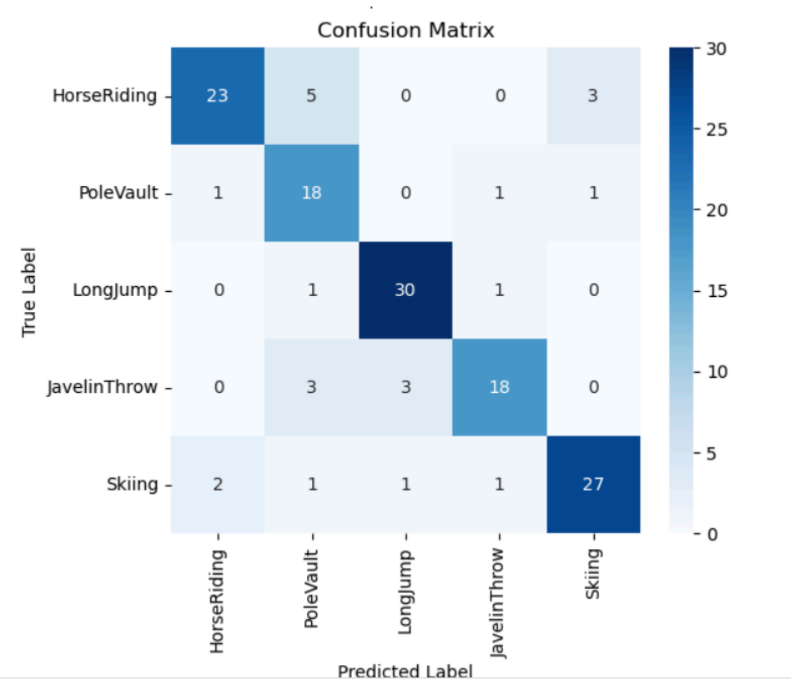
- Classification model is performing well, as all **AUC** values are close to 1.The **LongJump** and **Skiing** classes have the highest **AUC (0.97)**, indicating strong classification performance for these classes.

- The **HorseRiding** class has the lowest **AUC (0.95)**, but it's still performing very well.

3 CONFUSION MATRIX :

True Label	Predicted HorseRiding	Predicted PoleVault	Predicted LongJump	Predicted JavelinThrow	Predicted Skiing
HorseRiding	23	5	0	0	3
PoleVault	1	18	0	1	1
LongJump	0	1	30	1	0
JavelinThrow	0	3	3	18	0
Skiing	2	1	1	1	27

- **Diagonal values** show correct predictions, while **off-diagonal values** indicate misclassifications.



4 CLASSIFICATION REPORT TABLE:

CLASS	PRECISION	RECALL	F1-SCORE
HORSERIDING	0.88	0.74	0.81
POLEVAULT	0.64	0.86	0.73
LONGJUMP	0.88	0.94	0.91
JAVELINTHROW	0.86	0.75	0.80
SKIING	0.87	0.84	0.86

(i) Precision: Measures how many of the predicted positive cases were actually correct.

- Highest precision: HorseRiding (0.88), LongJump (0.88)

(ii) Recall: Measures how many actual positive cases were correctly identified.

- Highest recall: LongJump (0.94) (model captures most of this class correctly)

(iii) F1-Score: Harmonic mean of precision and recall, balancing both.

- Highest F1-Score: LongJump (0.91)

TESTING:

1 Testing with Input Data (Transformer + LSTM Hybrid):

```
    pred_prob = model.predict(features)
    pred_label = np.argmax(pred_prob)
    print("Predicted Action:", class_names[pred_label])

class_names = ["HorseRiding", "PoleVault", "LongJump", "JavelinThrow", "Skiing"]
process_video("/content/drive/MyDrive/DATA/JavelinThrow/v_JavelinThrow_g04_c02.avi", model, class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
Predicted Action: JavelinThrow
```

2 Testing with Input Data (Pre-Trained model):

```
def predict_video(video_path):
    video_tensor = preprocess_video(video_path)

    with torch.no_grad():
        outputs = model(video_tensor)
        predicted_class = torch.argmax(outputs, dim=1).item()

    print(f" Predicted Class: {label_map[predicted_class]}")

test_video_path = "/content/drive/MyDrive/DATA/HorseRiding/v_HorseRiding_g08_c02.avi"
predict_video(test_video_path)
```

```
Predicted Class: HorseRiding
```

3 Testing with YouTube Video :

Video Downloading: The project uses "**yt_dlp**" to download YouTube videos for analysis.
Frame Extraction: Key frames are sampled from the video to capture relevant motion sequences.
Preprocessing: Frames are resized, converted to grayscale, and normalized for model input.
Feature Extraction: CNN-based feature representations from each frame.
Action Classification: A trained deep learning model predicts the action category from extracted features.

```
def process_youtube_stream(youtube_url, model, class_names):
    frames = extract_frames_from_stream(youtube_url, num_frames=30)
    frames_resized = [cv2.resize(cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY), (96, 480)) for frame in frames]

    predictions = []
    for frame in frames_resized:
        frame = np.expand_dims(frame, axis=-1)
        frame = np.expand_dims(frame, axis=0)

        pred_prob = model.predict(frame)
        predictions.append(pred_prob)

    avg_pred = np.mean(predictions, axis=0)
    pred_label = np.argmax(avg_pred)

    print("Predicted Action:", class_names[pred_label])


class_names = ["HorseRiding", "PoleVault", "LongJump", "JavelinThrow", "Skiing"]
youtube_url = "https://youtube.com/shorts/dFzeqoPwhig?si=ttW1xs7bfqnylGyj"
process_youtube_stream(youtube_url, model, class_names)
```

Youtube link : https://youtube.com/shorts/dFzeqoPwhig?si=ttW1xs7bfqnylGyj

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 236ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 181ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 130ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 118ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 126ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 124ms/step
Predicted Action: HorseRiding
```

INITIAL APPROACH :

1st approach :

- Splitted them into train-test (80:20) , converted them into csv and followed on approach
- Reading from a CSV and then loading frames **one by one** adds overhead and **increased latency** during training.
- temporal information between frames **lost**, making it difficult to model video dynamics effectively.

2nd approach :

- converting all of them into 1D vectors
- But videos have both spatial (image-based) and temporal (motion-based) information. Flattening frames into 1D vectors discards spatial structure, making it difficult for the model to learn meaningful patterns.
- video frames are usually high-resolution images. Flattening them into 1D vectors results in extremely large feature spaces

3rd approach :

- extracted frames from videos, preprocessed them (resizing, normalization, and padding), and stored them as NumPy arrays for model training.
- Implemented functions to read videos from a dataset folder, process frames, and store them efficiently.

Challenges Faced:

- Missing class directories in the dataset. Some videos not yielding frames (handled with warnings) , Ensuring all video samples have a fixed number of frames via padding , Processing multiple formats (.avi) , Saving large processed datasets efficiently.

OUT OF CONTEXT :

7 days are huge to do this project but I am having 2 Online tests and 3 Interviews in this gap which made me packed in this week. However BIGVISION is my 1st choice , i kept an equal interval of time every day , which made me complete this assignment .

In my opinion further improvements can be **Finetuning** , deployment using **Fastapi** for backend and **Streamlit** for frontend (i am good with both of them) .