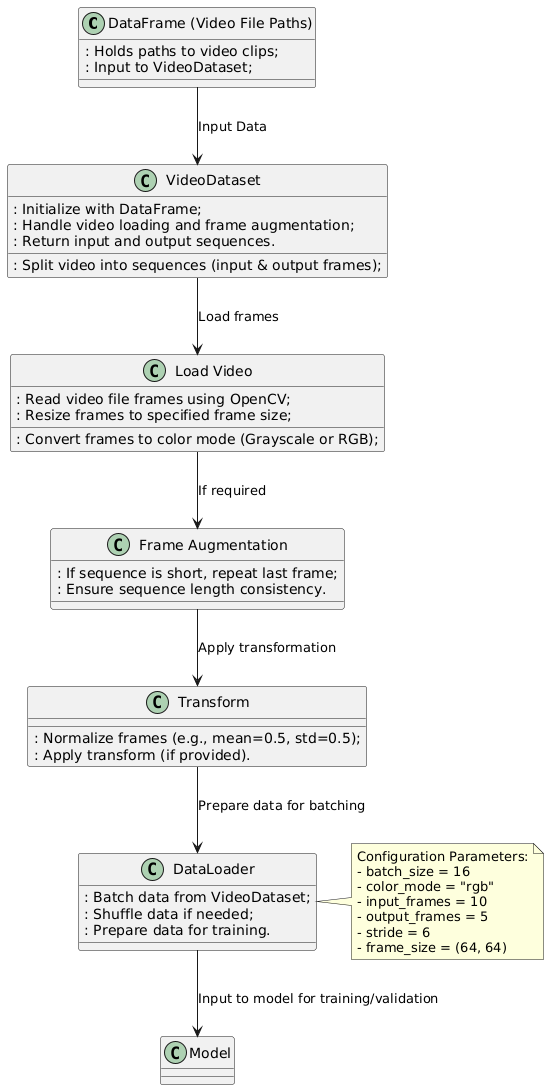
### Labels used:

### PENTA\_LABELS = ['BenchPress', 'Drumming', 'HorseRiding', 'Basketball', 'TennisSwing']

Data pipeline:



### Breakdown of the Process Flow:

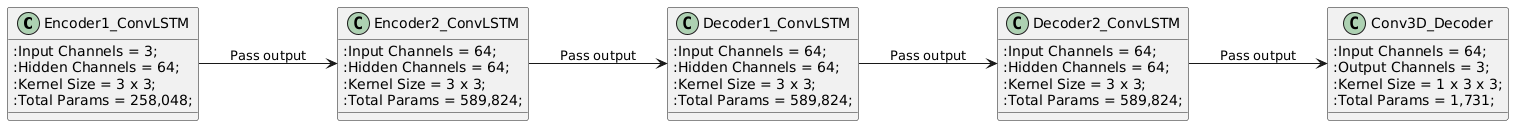
1. **DataFrame (Video File Paths)**:  
   The data frame stores paths to the video files that will be processed.
2. **VideoDataset**:  
   The VideoDataset class initializes with the DataFrame and splits the video files into sequences (input frames and output frames). It is responsible for preparing the data by calling methods like load\_video to load frames and augment\_frames to ensure sequences have consistent lengths.
3. **Load Video**:  
   This step involves reading frames from video files using OpenCV. The frames are resized and converted to the specified color mode (Grayscale or RGB).
4. **Frame Augmentation**:  
   If a video sequence is shorter than required, it is augmented by repeating the last frame to make the sequence the correct length.
5. **Transform**:  
   The frames are normalized (e.g., mean=0.5, std=0.5) or transformed as needed.
6. **DataLoader**:  
   After processing, the DataLoader batches the sequences, applies any necessary shuffling, and prepares the data for training or validation.
7. **Model**:  
   Finally, the batched data is fed into the model for training or inference.

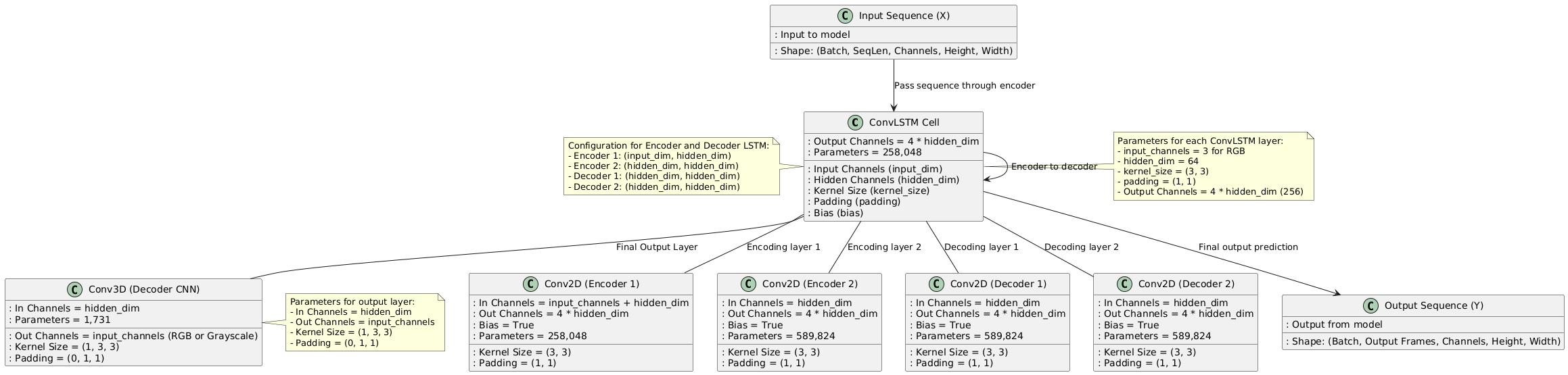
### Configuration Parameters:

A note on the side of the diagram shows key configuration parameters:

* batch\_size = 16
* color\_mode = "rgb"
* input\_frames = 10
* output\_frames = 5
* stride = 6
* frame\_size = (64, 64)

Conv Lstm





### Configuration:

* **Input Channels**: 3 (RGB input)
* **Hidden Dimension (nf)**: 64
* **Kernel Size**: (3, 3) for ConvLSTM and (1, 3, 3) for Conv3D
* **Stride**: 1 (default for ConvLSTM and Conv3D)
* **Padding**: (0, 1, 1) for Conv3D, (3//2, 3//2) for ConvLSTM

### 1. ****ConvLSTM (Encoder 1)****:

* **Input Channels**: 3 (RGB)
* **Hidden Channels**: 64
* **Kernel Size**: (3, 3)
* **Number of Gates**: 4 (i, f, o, g)

**Calculation**:

* Parameters per gate: (input\_channels + hidden\_channels) \* kernel\_height \* kernel\_width \* hidden\_channels
* For each gate:
  + Parameters per gate = (3 + 64) \* 3 \* 3 \* 64 = 64,512
* Total parameters for ConvLSTM = 64,512 \* 4 gates = **258,048 parameters**

### 2. ****ConvLSTM (Encoder 2)****:

* **Input Channels**: 64 (from Encoder 1 output)
* **Hidden Channels**: 64
* **Kernel Size**: (3, 3)
* **Number of Gates**: 4 (i, f, o, g)

**Calculation**:

* Parameters per gate: (64 + 64) \* 3 \* 3 \* 64 = 147,456
* Total parameters for ConvLSTM = 147,456 \* 4 gates = **589,824 parameters**

### 3. ****ConvLSTM (Decoder 1)****:

* **Input Channels**: 64 (from Encoder 2 output)
* **Hidden Channels**: 64
* **Kernel Size**: (3, 3)
* **Number of Gates**: 4 (i, f, o, g)

**Calculation**:

* Parameters per gate: (64 + 64) \* 3 \* 3 \* 64 = 147,456
* Total parameters for ConvLSTM = 147,456 \* 4 gates = **589,824 parameters**

### 4. ****ConvLSTM (Decoder 2)****:

* **Input Channels**: 64 (from Decoder 1 output)
* **Hidden Channels**: 64
* **Kernel Size**: (3, 3)
* **Number of Gates**: 4 (i, f, o, g)

**Calculation**:

* Parameters per gate: (64 + 64) \* 3 \* 3 \* 64 = 147,456
* Total parameters for ConvLSTM = 147,456 \* 4 gates = **589,824 parameters**

### 5. ****Conv3D (Decoder Output)****:

* **Input Channels**: 64 (from Decoder 2 output)
* **Output Channels**: 3 (to match the input channels for RGB)
* **Kernel Size**: (1, 3, 3)
* **Padding**: (0, 1, 1)

**Calculation**:

* Parameters = (input\_channels \* kernel\_depth \* kernel\_height \* kernel\_width \* output\_channels) + (output\_channels if bias else 0)
  + Here kernel\_depth = 1 (for 3D), kernel\_height = 3, kernel\_width = 3.
* Parameters = (64 \* 1 \* 3 \* 3 \* 3) = 1,728
* Plus bias (if used): 3 (for the 3 output channels)
* **Total parameters for Conv3D = 1,728 + 3 = 1,731 parameters**

### ****Total Parameters Calculation****:

* **Encoder 1 ConvLSTM**: 258,048 parameters
* **Encoder 2 ConvLSTM**: 589,824 parameters
* **Decoder 1 ConvLSTM**: 589,824 parameters
* **Decoder 2 ConvLSTM**: 589,824 parameters
* **Conv3D Decoder**: 1,731 parameters

**Total Parameters** = 258,048 + 589,824 + 589,824 + 589,824 + 1,731 = **2,029,251 parameters**

### 1. ****ConvLSTM Cell Layer****

* **Purpose**: The heart of the architecture, this layer integrates convolution and LSTM (Long Short-Term Memory) cells, allowing the model to capture both spatial and temporal dependencies in the input sequence of frames.
* **Parameters**:
  + **Input Channels (**input\_dim**)**: 3 for RGB input images.
  + **Hidden Channels (**hidden\_dim**)**: 64 (the hidden dimension controls the capacity of the network to remember spatial features over time).
  + **Kernel Size (**kernel\_size**)**: (3, 3) for the convolutional operation.
  + **Padding**: (1, 1) ensures the spatial dimensions remain the same after applying the kernel.
  + **Output Channels**: 4 \* hidden\_dim = 256 because the ConvLSTM cell has 4 gates: input gate, forget gate, output gate, and cell state.
* **Working**:
  + The ConvLSTMCell operates on a sequence of frames (input sequence), where each frame is processed through the LSTM mechanism to update the hidden and cell states. These states are used to remember important features across time steps in the video sequence.

### 2. ****Encoder Layers (Conv2D)****

There are two encoder layers, each applying a convolutional operation followed by the LSTM mechanism.

#### a) ****Encoder 1 (Conv2D Layer)****

* **Input**: The input to this layer is the combination of the current frame and the previous hidden state.
* **Channels**: The input channels are the sum of the input channels (input\_channels = 3 for RGB) and the hidden state channels (hidden\_dim = 64), so the input channels become 3 + 64 = 67.
* **Output**: The output is 4 \* hidden\_dim = 256 channels, corresponding to the number of gates in the ConvLSTMCell.
* **Kernel Size**: (3, 3) with padding (1, 1) ensures spatial resolution is preserved.
* **Output Channels**: 256 (since 4 \* hidden\_dim = 256).

#### b) ****Encoder 2 (Conv2D Layer)****

* **Input**: This layer receives the output of Encoder 1 (which has hidden\_dim = 64 channels).
* **Output**: The output is also 4 \* hidden\_dim = 256 channels, similar to Encoder 1.
* **Kernel Size**: (3, 3) with padding (1, 1) to maintain spatial dimensions.
* **Output Channels**: 256.

### 3. ****Decoder Layers (Conv2D)****

Similar to the encoder, the decoder has two layers of ConvLSTM followed by a final 3D convolution layer to decode the sequence of frames.

#### a) ****Decoder 1 (Conv2D Layer)****

* **Input**: This layer receives the output from Encoder 2, which has hidden\_dim = 64 channels.
* **Output**: The output is 4 \* hidden\_dim = 256 channels.
* **Kernel Size**: (3, 3) with padding (1, 1).
* **Output Channels**: 256.

#### b) ****Decoder 2 (Conv2D Layer)****

* **Input**: This layer receives the output from Decoder 1.
* **Output**: Again, the output is 4 \* hidden\_dim = 256 channels.
* **Kernel Size**: (3, 3) with padding (1, 1).

### 4. ****Decoder CNN (Conv3D Layer)****

* **Purpose**: This is the final output layer, which is a 3D convolutional layer that decodes the output sequence back into the original number of input channels (RGB in this case).
* **Input**: The input to this layer is the final output from Decoder 2, which has hidden\_dim = 64 channels.
* **Output**: The output is input\_channels = 3, matching the input channels of the sequence (RGB).
* **Kernel Size**: (1, 3, 3) with padding (0, 1, 1) to ensure spatial dimensions match the input.
* **Output Channels**: 3 (the number of channels for RGB images).

### 5. ****Flow of Data Through Layers****

1. **Input Sequence (X)**:
   * The input sequence consists of a batch of frames from a video, with shape (Batch, SeqLen, Channels, Height, Width). For example, with RGB input, Channels = 3.
2. **Encoding**:
   * The sequence of frames passes through the **Encoder 1** and **Encoder 2** layers. These layers use ConvLSTM cells to capture spatial and temporal dependencies and generate hidden representations that summarize the sequence up to the current frame.
3. **Decoding**:
   * After encoding, the decoder layers (Decoder 1 and Decoder 2) take the encoded sequence and predict future frames. The hidden states from the encoder are used as the initial input to the decoder layers.
4. **Output Generation**:
   * The final output is generated by the **Conv3D Decoder CNN** layer, which reconstructs the output sequence from the decoder’s hidden state back to the original number of channels (e.g., 3 for RGB frames).
   * The output shape is (Batch, Output Frames, Channels, Height, Width), where the number of frames generated corresponds to the prediction horizon (e.g., 5 frames).

### 6. ****Training Objective****

* **Loss Calculation**: During training, the model minimizes a combined loss function, which consists of two components:
  1. **MSE Loss**: Measures the difference between the predicted and actual frames in terms of pixel-wise error.
  2. **SSIM Loss**: Measures the structural similarity between the predicted and actual frames, helping to preserve texture and perceptual details.
* **Combined Loss**: The total loss is a weighted combination of the MSE and SSIM values. The formula used is:  Loss Loss ScoreCombined Loss=MSE Loss−0.5×SSIM Score

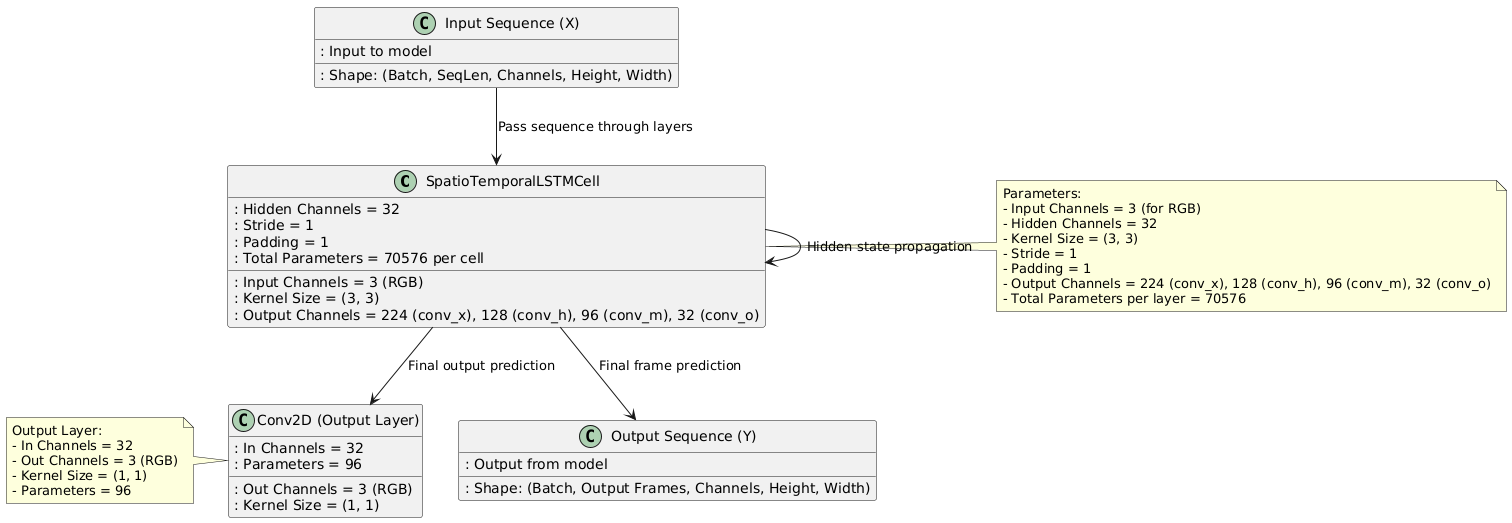
### 7. ****Overall Architecture Summary****

* **Layer-wise Architecture**:
  1. **Input**: Video frames (RGB or grayscale) of size (Batch, SeqLen, Channels, Height, Width).
  2. **Encoder**: Two ConvLSTM layers for encoding the input sequence and extracting spatiotemporal features.
  3. **Decoder**: Two ConvLSTM layers for predicting the future frames.
  4. **Final Layer**: A Conv3D layer to reconstruct the predicted sequence in the same format as the input.
* **Training**: The model is trained using a combined loss of MSE and SSIM to minimize pixel-wise and perceptual errors simultaneously.

### 8. ****Configuration Summary****:

* **Input Channels**: 3 (for RGB)
* **Hidden Channels**: 64 (for both Encoder and Decoder)
* **Kernel Size**: (3, 3) for convolution operations
* **Output Channels**: 256 (for each ConvLSTM cell)
* **Padding**: (1, 1) for Conv2D operations
* **Final Output Channels**: 3 (RGB channels)

PredRNN:



### 1. ****Parameter Calculation****

#### For the SpatioTemporalLSTMCell Class:

* **Input Channels**: 3 (for RGB input)
* **Number of Hidden Units**: 32 for each layer (based on the example configuration [32, 32, 32]).
* **Filter Size (Kernel Size)**: 3
* **Stride**: 1
* **Padding**: 1

Each SpatioTemporalLSTMCell layer uses convolution to handle both spatial and temporal information, and for each LSTM cell, the parameters are calculated as follows:

* conv\_x **(input to LSTM cell)**:
  + Output channels: num\_hidden \* 7 = 32 \* 7 = 224
  + Parameters for conv\_x: Parameters=(in\_channels×out\_channels×kernel\_height×kernel\_width)+bias  bias as bias=False)Parameters=(3×224×3×3)=2016(without bias as bias=False)
* conv\_h **(hidden state to LSTM cell)**:
  + Output channels: num\_hidden \* 4 = 32 \* 4 = 128
  + Parameters for conv\_h: Parameters=(num\_hidden×out\_channels×kernel\_height×kernel\_width)=(32×128×3×3)=36864
* conv\_m **(memory to LSTM cell)**:
  + Output channels: num\_hidden \* 3 = 32 \* 3 = 96
  + Parameters for conv\_m: Parameters=(num\_hidden×out\_channels×kernel\_height×kernel\_width)=(32×96×3×3)=27648
* conv\_o **(output to LSTM cell)**:
  + Output channels: num\_hidden = 32
  + Parameters for conv\_o: Parameters=(num\_hidden×num\_hidden×kernel\_height×kernel\_width)=(64×32×1×1)=2048
* **Total Parameters for Each SpatioTemporalLSTMCell**:

 Parameters bias)Total Parameters=2016+36864+27648+2048=70576(without bias)

#### For the PredRNN Class:

* **Number of Layers**: 3
* **Input Channels**: 3 (RGB)
* **Hidden Channels**: [32, 32, 32]
* **Input Frames**: 10
* **Output Frames**: 5

For each SpatioTemporalLSTMCell layer, the parameters are as calculated above. For the entire PredRNN model:

* **Parameters for LSTM layers**:
  + Number of LSTM cells: 3 (one for each layer, and for each layer, we already calculated 70576 parameters).
  + Total parameters for LSTM layers: 3×70576=211728
* **Output Layer (Conv2D)**:
  + The final output layer is a 1x1 convolution to map the final hidden state back to the input channels (RGB).
  + Output channels: 3
  + Parameters for the output layer: Parameters=(num\_hidden×output\_channels×kernel\_height×kernel\_width)=(32×3×1×1)=96
* **Total Parameters for** PredRNN:

 Parameters layers layerTotal Parameters=211728(LSTM layers)+96(output layer)=211824

Thus, the total number of parameters in the PredRNN model is **211,824**.

### ****Explanation of the**** PredRNN ****Architecture****

The PredRNN model is designed for **future frame prediction** in sequences of images or videos. It uses **SpatioTemporal LSTM cells**, which combine the benefits of both **spatial** and **temporal** modeling. Here’s a detailed breakdown of the architecture:

### ****1. Layers and Structure****

#### ****SpatioTemporal LSTM Cells****

The model consists of **3 layers** of **SpatioTemporal LSTM Cells**, and each layer processes the input sequence in a hierarchical manner. These layers are responsible for encoding and decoding the temporal dependencies, as well as capturing the spatial features of the frames.

* **Number of Layers**:
  + The model has 3 layers of LSTM cells. Each layer processes the sequence of frames and updates the hidden states at each time step.
* **Hidden Units in Each Layer**:
  + The number of hidden units in each LSTM layer is set to **32** (based on the configuration [32, 32, 32]).

#### ****Layer Composition****:

Each SpatioTemporal LSTMCell layer is composed of:

* conv\_x: Convolutional layer to process the input sequence.
* conv\_h: Convolutional layer for the hidden state.
* conv\_m: Convolutional layer for the memory state.
* conv\_o: Convolutional layer for output gating.

These convolutional layers process both spatial (image-based) and temporal (sequence-based) dependencies at the same time, making the SpatioTemporal LSTMCell ideal for video frame prediction.

* **Encoding**:
  + The first part of the sequence (10 frames in this case) is passed through the SpatioTemporal LSTMCell layers.
  + The cells update the hidden and memory states over time to capture both the temporal dynamics (how the sequence progresses) and spatial information (the appearance of each frame).
* **Decoding**:
  + Once the encoding is complete, the model begins to predict the future frames (5 frames in this case). The final hidden state after encoding is passed through the LSTM layers again to generate the output frames one by one.

#### ****Output Layer****

* After processing through the LSTM cells, the last hidden state is passed through a **1x1 Convolutional Layer** (conv\_last) to map the final hidden state back to the RGB output space (i.e., from 32 channels to 3 channels). This is the predicted future frame for each timestep.

### ****2. Flow of Data****

Here’s a detailed flow of data through the PredRNN model:

1. **Input Sequence**:
   * The model takes a sequence of input frames (e.g., 10 frames) in the form of a tensor with shape (Batch Size, SeqLen, Channels, Height, Width) where:
     + Batch Size is the number of images processed together.
     + SeqLen is the length of the input sequence (10 frames).
     + Channels is the number of channels in each frame (3 for RGB images).
     + Height and Width are the spatial dimensions of the frames (e.g., 64x64).
2. **Encoding**:
   * The sequence is passed through the **SpatioTemporal LSTM cells**. In each LSTM cell, the input frame and the hidden state from the previous time step are processed to update the memory and hidden states.
   * The hidden state after each layer is passed to the next layer, capturing deeper temporal and spatial dependencies as the sequence progresses.
3. **Prediction**:
   * After encoding, the model begins to predict future frames. For each future frame, the output of the last LSTM layer is passed through the conv\_last layer to produce a frame.
   * This prediction is done for output\_frames (5 in this case), and each frame is produced sequentially.
4. **Final Output**:
   * The predicted frames are stacked to form the output sequence, which has the shape (Batch Size, Output Frames, Channels, Height, Width).

### ****3. Number of Parameters****

For each SpatioTemporalLSTMCell layer:

* **Conv2D layers** are used to compute the gates and update states. These include:
  + conv\_x (input-to-hidden),
  + conv\_h (hidden-to-hidden),
  + conv\_m (memory-to-hidden),
  + conv\_o (output).
* The total number of parameters for each SpatioTemporal LSTMCell is calculated to be **70,576** (without bias).

For the entire PredRNN model:

* **3 layers** of LSTM cells, each with 70,576 parameters, result in a total of **211,728 parameters** for the LSTM layers.
* The **output layer** adds an additional **96 parameters** (since it’s a 1x1 convolution from 32 channels to 3 channels).
* The total number of parameters for the entire PredRNN model is **211,824**.

### ****4. Training and Metrics****

* **Loss Functions**:
  + The model is trained using **MSE Loss (Mean Squared Error)**, which measures the pixel-wise error between the predicted frames and the target frames.
  + **SSIM (Structural Similarity Index)** is also used as a metric to evaluate perceptual quality. SSIM measures how similar the predicted frame is to the target frame in terms of structure, luminance, and texture.
* **Training Process**:
  + The training process involves feeding the model with an input sequence of frames, performing forward passes, calculating the losses, and then updating the model parameters using backpropagation.
  + **Adam Optimizer** is used for optimization, and a **ReduceLROnPlateau Scheduler** is employed to adjust the learning rate during training to prevent overfitting.

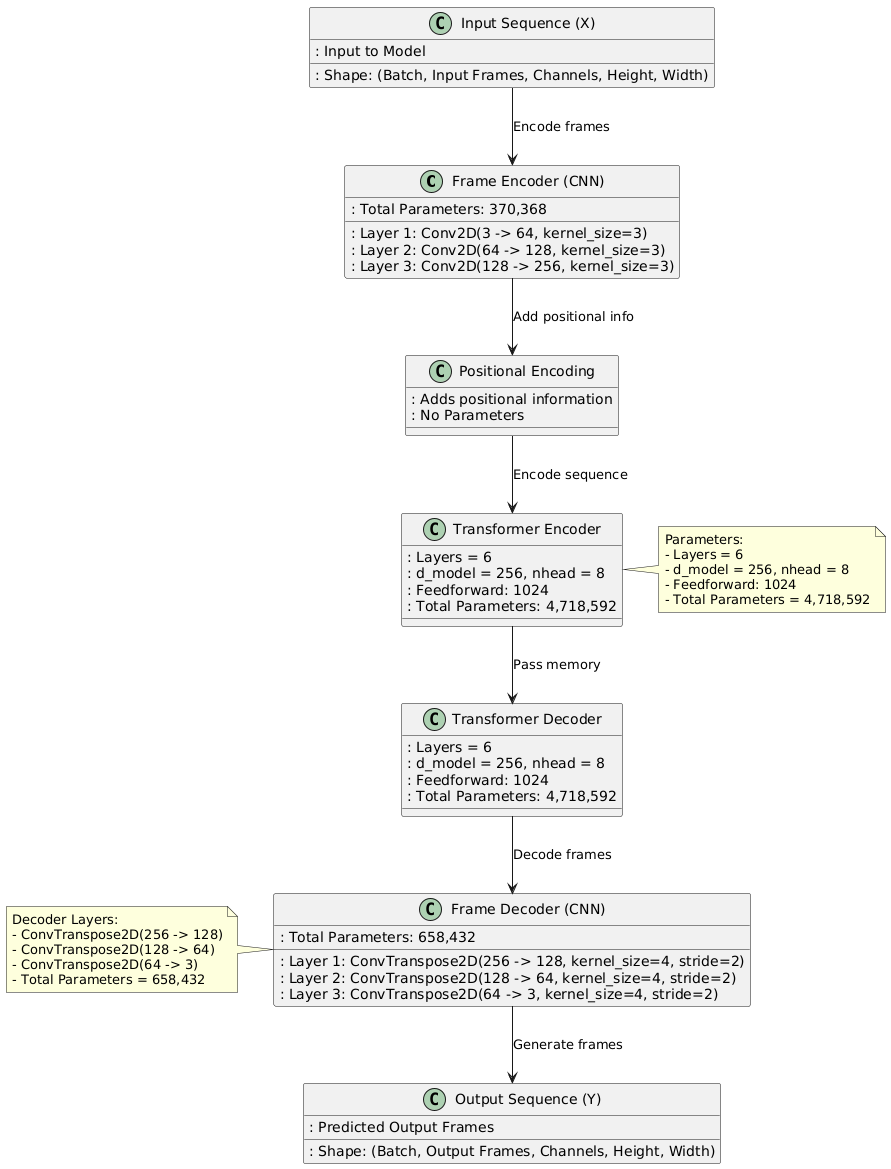
### ****5. Summary of Layers****

* **Input Sequence**:
  + Shape: (Batch, SeqLen, Channels, Height, Width).
* **SpatioTemporal LSTM Cells**:
  + 3 layers, each with hidden channels of 32.
  + Each cell processes both spatial and temporal dependencies of input frames.
* **Output Layer**:
  + A final Conv2D layer (1x1 convolution) produces the output frames (RGB).
* **Total Parameters**:
  + **211,824** parameters in total, with 3 LSTM layers and 1 output layer.

### ****Conclusion****

The PredRNN model is an effective architecture for future frame prediction in videos. It leverages SpatioTemporal LSTM cells to capture both the spatial structure of images and the temporal dynamics of video sequences, making it suitable for tasks such as video forecasting, action prediction, and motion modeling. The model's ability to predict future frames is driven by its deep hierarchical structure, where each layer progressively captures higher-level temporal and spatial features.

Transformer:



### ****Parameter Calculation for**** VideoTransformer

We calculate the number of parameters for each layer in the model:

#### ****1. Frame Encoder (CNN Encoder for each frame)****:

* **Layer 1**: Conv2D(input\_channels=3, out\_channels=64, kernel\_size=3, padding=1)
  + Parameters = 3×64×3×3=1,728 (no bias as bias=False).
* **Layer 2**: Conv2D(input\_channels=64, out\_channels=128, kernel\_size=3, padding=1)
  + Parameters = 64×128×3×3=73,728.
* **Layer 3**: Conv2D(input\_channels=128, out\_channels=256, kernel\_size=3, padding=1)
  + Parameters = 128×256×3×3=294,912.
* **Total Parameters for Frame Encoder**:

1,728+73,728+294,912=370,368

#### ****2. Positional Encoding****:

* **No parameters** as it uses precomputed sinusoidal functions.

#### ****3. Transformer Encoder****:

* **TransformerEncoderLayer**:
  + **Multi-head Attention**:
    - For dmodel​=256,nhead​=8:
    - Query, Key, and Value matrices: 256×256×3=196,608
    - Output projection: 256×256=65,536.
    - Total parameters for attention: 196,608+65,536=262,144.
  + **Feedforward Layer**:
    - Input to hidden: 256×1024=262,144.
    - Hidden to output: 1024×256=262,144.
    - Total parameters for feedforward: 262,144+262,144=524,288.
  + **Layer Norm**: Negligible.
  + **Total Parameters for One Encoder Layer**: 262,144+524,288=786,432.
* **6 Encoder Layers**: 786,432×6=4,718,592.

#### ****4. Transformer Decoder****:

* Similar to the encoder, each TransformerDecoderLayer has:
  + **Multi-head Attention**: 262,144.
  + **Feedforward Layer**: 524,288.
* **Total Parameters for One Decoder Layer**: 262,144+524,288=786,432.
* **6 Decoder Layers**: 786,432×6=4,718,592.

#### ****5. Frame Decoder (CNN Decoder for each frame)****:

* **Layer 1**: ConvTranspose2D(input\_channels=256, out\_channels=128, kernel\_size=4, stride=2, padding=1)
  + Parameters = 256×128×4×4=524,288.
* **Layer 2**: ConvTranspose2D(input\_channels=128, out\_channels=64, kernel\_size=4, stride=2, padding=1)
  + Parameters = 128×64×4×4=131,072.
* **Layer 3**: ConvTranspose2D(input\_channels=64, out\_channels=3, kernel\_size=4, stride=2, padding=1)
  + Parameters = 64×3×4×4=3,072.
* **Total Parameters for Frame Decoder**:

524,288+131,072+3,072=658,432

#### ****6. Total Parameters****:

 Encoder:  Encoder:  Decoder:  Decoder:  Frame Encoder: 370,368Transformer Encoder: 4,718,592Transformer Decoder: 4,718,592Frame Decoder: 658,432Total: 10,466,384parameters.

### ****Explanation of the VideoTransformer Model****

The VideoTransformer is a deep learning model designed for video prediction tasks. It uses a combination of convolutional neural networks (CNNs) and transformers to capture both spatial and temporal information. Here's an overview of how the model works, the layers used, and its output:

### ****1. Input Layer:****

* **Input Shape**: The model accepts a sequence of video frames as input, represented as a 5D tensor: (batch\_size, num\_frames, channels, height, width).
  + Example: If you input 10 frames with RGB channels of size 64x64, the shape will be (batch\_size, 10, 3, 64, 64).

### ****2. Frame Encoder (CNN) - Spatial Feature Extraction:****

* The **frame encoder** extracts spatial features from each individual video frame. This is done using 3 convolutional layers.
  + **Layer 1**:
    - Conv2D(3 -> 64, kernel\_size=3, stride=1, padding=1): The first layer converts the input channels (RGB, 3 channels) to 64 channels using a kernel size of 3x3.
  + **Layer 2**:
    - Conv2D(64 -> 128, kernel\_size=3, stride=1, padding=1): The second layer increases the feature depth to 128 channels.
  + **Layer 3**:
    - Conv2D(128 -> 256, kernel\_size=3, stride=1, padding=1): The third layer increases the depth to the final d\_model of 256 channels.

After the convolution layers, the frames are downsampled using **MaxPooling** layers, reducing spatial dimensions (height and width). The output of the frame encoder for each frame is a feature map of shape (batch\_size, seq\_len, d\_model, reduced\_height, reduced\_width).

### ****3. Positional Encoding (Transformer) - Temporal Information:****

* Since transformers don’t have an inherent notion of sequence order, **positional encoding** is added to the feature maps to encode the relative position of each frame in the sequence.
  + The positional encoding is based on sinusoidal functions, which allows the model to understand the order of frames.

### ****4. Transformer Encoder - Temporal Feature Extraction:****

* The transformer encoder uses **multi-head self-attention** and **feedforward layers** to model temporal relationships between frames.
  + **Number of Encoder Layers**: 6.
  + **d\_model**: 256 (the depth of each feature representation).
  + **nhead**: 8 (number of attention heads).
  + **dim\_feedforward**: 1024 (dimension of the feedforward network).

The encoder processes the entire sequence of frames (with positional encodings) and generates a memory representation. The attention mechanism helps capture long-range dependencies between frames, enabling the model to predict future frames based on past frames.

### ****5. Transformer Decoder - Frame Prediction:****

* The decoder takes the **memory** from the transformer encoder and a sequence of **start tokens** (initialized to zeros) as input. It generates the predicted future frames sequentially.
  + The decoder uses **multi-head self-attention** and **cross-attention** mechanisms to generate new frames based on the encoder's output.
  + **Number of Decoder Layers**: 6, similar to the encoder.
  + The decoder's output for each frame is a latent representation of the frame's features.

### ****6. Frame Decoder (CNN) - Reconstructing Frames:****

* The **frame decoder** is a **deconvolutional network (transposed convolution)** that reconstructs the predicted frame from the latent features output by the transformer decoder.
  + **Layer 1**: ConvTranspose2D(d\_model -> 128, kernel\_size=4, stride=2, padding=1): Upsamples the latent features back to a higher resolution.
  + **Layer 2**: ConvTranspose2D(128 -> 64, kernel\_size=4, stride=2, padding=1): Further upsamples the features.
  + **Layer 3**: ConvTranspose2D(64 -> 3, kernel\_size=4, stride=2, padding=1): Finally, reconstructs the output frame with the same number of channels as the input (3 for RGB).

The output of this block is a sequence of predicted frames.

### ****7. Output:****

* The output of the model is a sequence of predicted frames: (batch\_size, output\_frames, channels, height, width). For example, the model could predict 5 future frames based on 10 input frames, where each predicted frame is of size 64x64 with 3 channels (RGB).
  + The **output frames** are reconstructed using the frame decoder from the latent representations produced by the transformer.

### ****Summary of Layers and Structure:****

1. **Frame Encoder**: 3 convolutional layers.
2. **Positional Encoding**: Adds temporal information.
3. **Transformer Encoder**: 6 layers of self-attention and feedforward networks.
4. **Transformer Decoder**: 6 layers of self-attention and cross-attention.
5. **Frame Decoder**: 3 deconvolutional layers to generate output frames.

### ****Key Features:****

* **Spatial Feature Extraction**: The CNN-based frame encoder extracts important spatial features from each frame.
* **Temporal Modeling**: The transformer encoder models long-range temporal dependencies between frames.
* **Frame Generation**: The transformer decoder predicts future frames, which are then reconstructed using deconvolutional layers.

This combination of CNNs and transformers allows the model to capture both spatial patterns within individual frames and temporal relationships between them, making it effective for tasks like video prediction and frame interpolation.

Testing mse and ssim convlstm:

=== Overall Results ===

Total videos processed: 117

Metrics by class:

mse ssim

mean std mean std

label

Basketball 0.010996 0.010695 0.713141 0.208380

BenchPress 0.009430 0.004686 0.793808 0.077303

Drumming 0.008988 0.005364 0.758777 0.130059

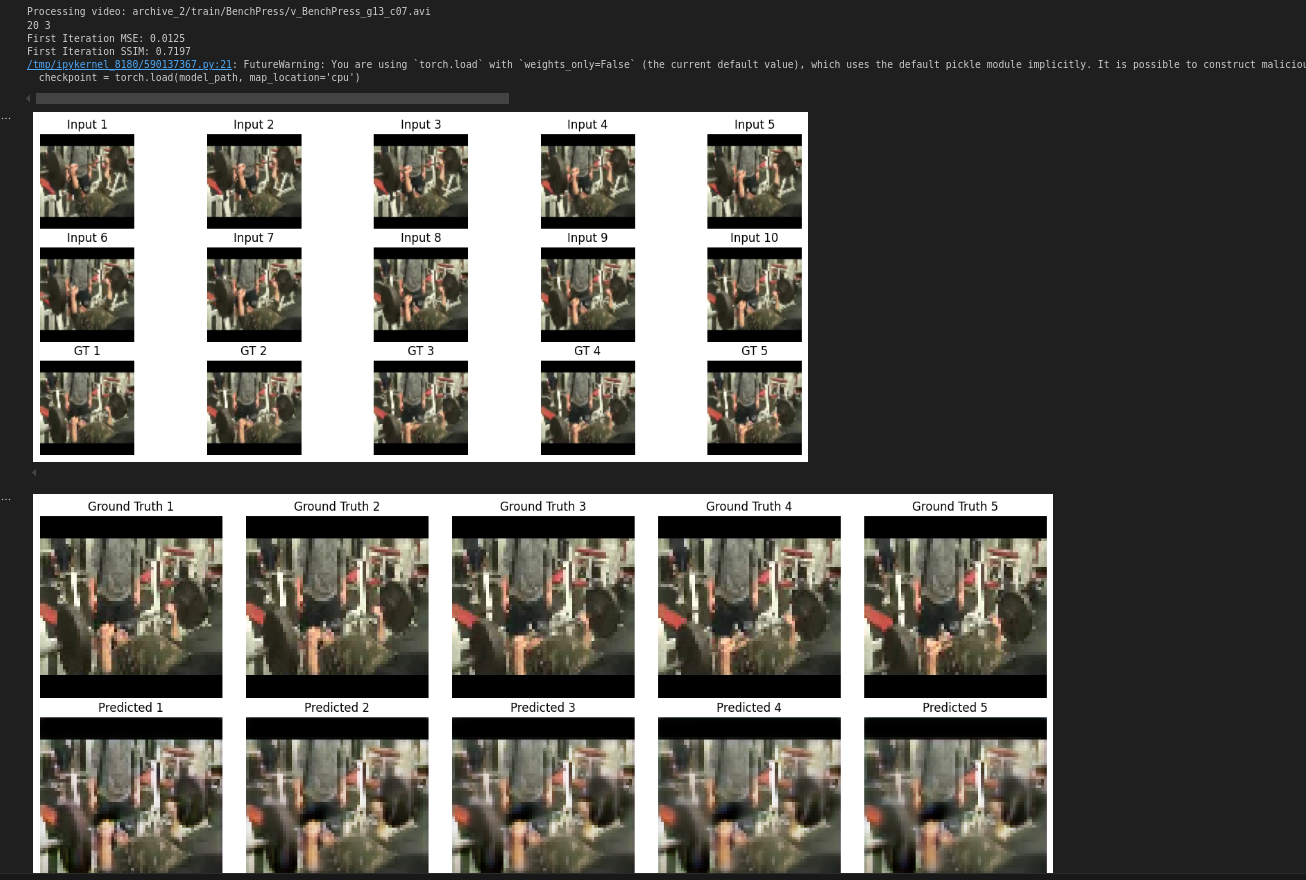
HorseRiding 0.010456 0.006641 0.621030 0.164910

TennisSwing 0.008478 0.012923 0.795530 0.204559

Overall metrics:

Average MSE: 0.0098 ± 0.0089

Average SSIM: 0.7334 ± 0.1789



Testing mse and ssim predrnn:

=== Overall Results ===

Total videos processed: 117

Metrics by class:

mse ssim

mean std mean std

label

Basketball 0.008777 0.008757 0.726517 0.198731

BenchPress 0.008135 0.003809 0.787692 0.076165

Drumming 0.008289 0.004236 0.755830 0.128249

HorseRiding 0.008440 0.005079 0.626892 0.153441

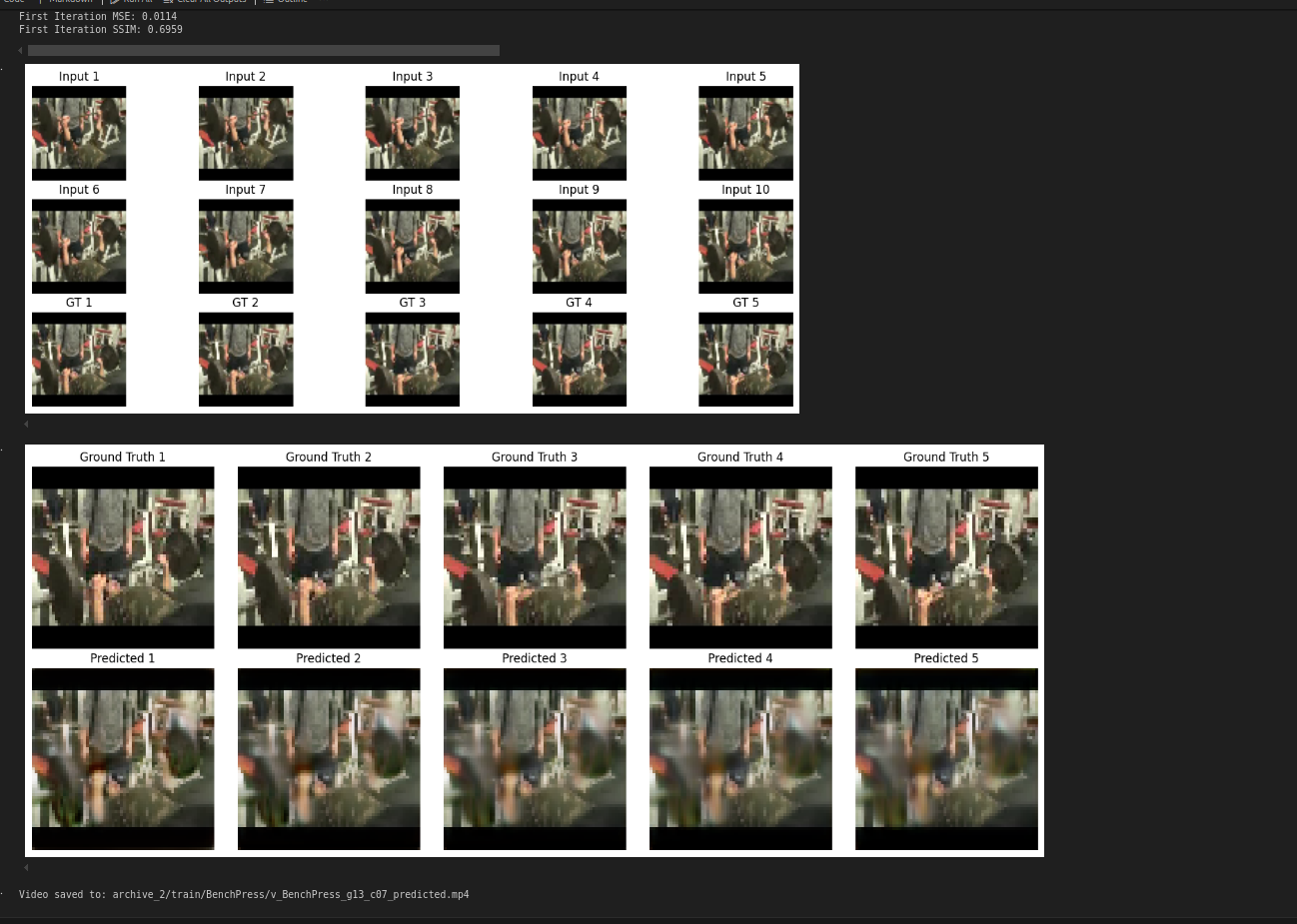
TennisSwing 0.007497 0.011778 0.791787 0.208757

Overall metrics:

Average MSE: 0.0083 ± 0.0075

Average SSIM: 0.7361 ± 0.1728

Detailed results saved to: test\_results.csv



Testing mse and ssim transformers:

=== Overall Results ===

Total videos processed: 117

Metrics by class:

mse ssim

mean std mean std

label

Basketball 0.224433 0.080394 0.007725 0.009149

BenchPress 0.190018 0.079595 0.008347 0.011180

Drumming 0.149640 0.102954 0.040368 0.067772

HorseRiding 0.277837 0.117567 0.001566 0.003596

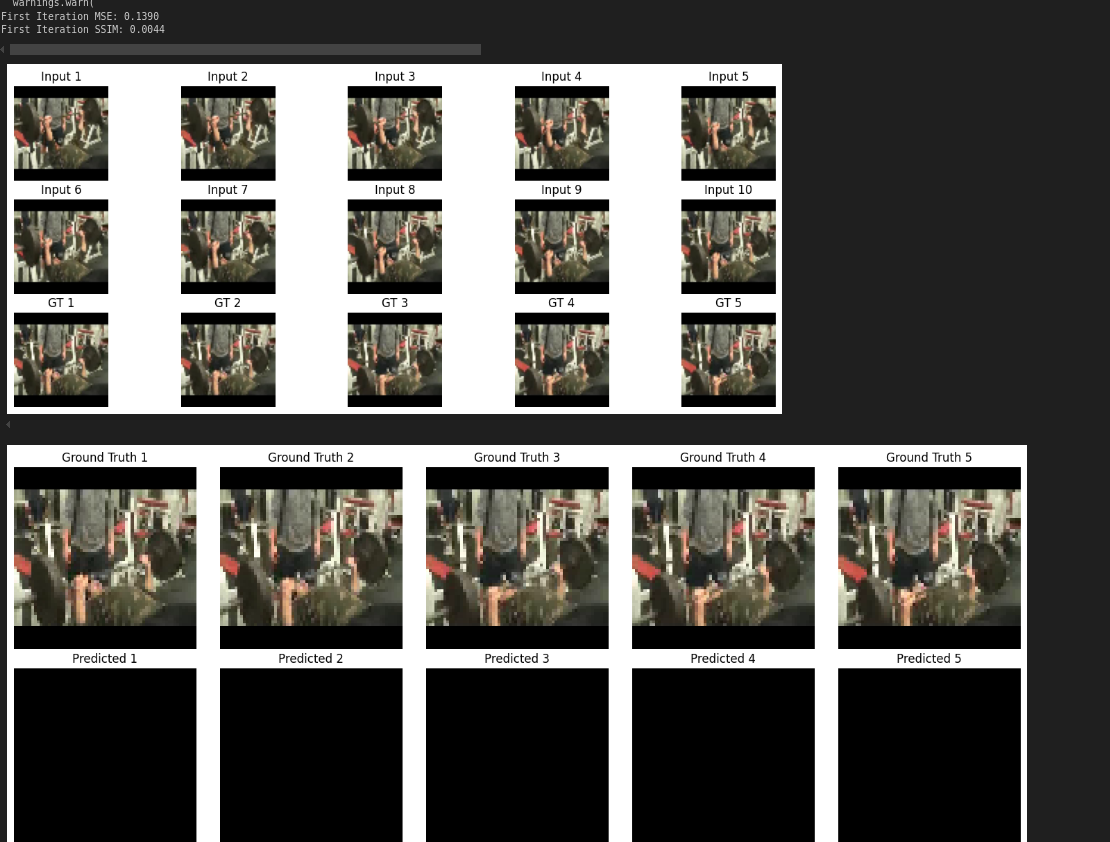
TennisSwing 0.242490 0.072117 0.011344 0.032446

Overall metrics:

Average MSE: 0.2180 ± 0.0987

Average SSIM: 0.0132 ± 0.0345

Detailed results saved to: test\_results.csv



Front end:

