📌 Introduction

This project presents an Action Recognition System that uses a 3D Convolutional Neural Network (3D CNN) built with Keras and OpenCV to classify different human actions from video clips. The model is designed to recognize five specific actions: Boxing, HandWaving, Jogging, Running, and Walking.

To enhance usability and testing, a few sample input videos have been included in the GitHub repository. These videos help demonstrate how the model processes and classifies actions effectively. Additionally, the model is capable of accepting real-time input from a live camera, allowing for dynamic action recognition directly from webcam feeds.

Input:

- Loads a video from the specified path.
- Resizes each frame to 16x16 pixels.
- Converts to grayscale.
- Prepares input in the shape expected by the model: (1, 1, 16, 16, 15).

🛠 Preprocessing

- **Step 1:** Extract frames from each video using **OpenCV**.
- **Step 2:** Resize frames to **16x16 pixels** and convert them to **grayscale**.
- **Step 3:** Store the frames as a **3D array** with dimensions (15, 16, 16).
- **Step 4:** Normalize pixel values to the range [0, 1].

🏗 Model Architecture

- **Model Type:** 3D Convolutional Neural Network (3D CNN).
- **Layers Used:**
    - Conv3D for spatial-temporal feature extraction.
    - MaxPooling3D for dimensionality reduction.

    - Dropout for preventing overfitting.
    - Dense layers for classification.

- Input: (1, 16, 16, 15)

- Conv3D → MaxPooling3D → Dropout → Flatten → Dense → Softmax

Architecture:

[Input Video]

↓

[Motion Detection]

↓

[Is Motion Detected?] → No → [Loop back to Motion Detection]

↓

Yes

↓

[Human Detection]

↓

[Multiple Tracking]

↓

[Convolutional Layer]

↓

[Pooling Layer]

↓

[Fully Connected Layer]

↓

[Output]

Building the 3D CNN Model:

```
model = Sequential()

model.add(Convolution3D((32,32), 16, 16, 16, activation='relu', input_shape=(128,128,3), padding='same'))

model.add(MaxPooling3D((3,3,3)))

model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(nb_classes, activation='softmax'))
```

- **Layers Used:**
    - **Convolution3D:** Extracts spatial and temporal features.
    - **MaxPooling3D:** Reduces spatial dimensions.
    - **Dropout:** Prevents overfitting.
    - **Flatten and Dense Layers:** Converts 3D features to 1D and classifies them.

🏋️ Training the Model

- **Batch Size:** 2
- **Epochs:** 50
- **Optimizer:** RMSprop
- **Loss Function:** Categorical Cross-Entropy
- **Metrics:** Accuracy

Training Command:

```
model.fit(X_train, Y_train, batch_size=2, epochs=50, validation_split=0.3)
```

🎯 Predicting Actions

Step 1: Prepare a new video as input.

Step 2: Run prediction:

```
predicted_class = model.predict(input_video)

predicted_label = np.argmax(predicted_class)
```

Step 3: Interpret the output:

```
label_map = {
    0: 'Boxing',
    1: 'HandWaving',
    2: 'Jogging',
    3: 'Running',
    4: 'Walking'
}
print("Predicted Action:", label_map[predicted_label])
```

Sample Output:

```
Predicted Action: Jogging
```

🎯 Making Predictions

Step 1: Prepare a new video as input.

Step 2: Run prediction:

```
# Ensure the input is float32 and normalized
input_video = input_video.reshape(1, 1, img_rows, img_cols, img_depth)
input_video -= np.mean(input_video)
input_video /= np.max(input_video)

# Predict the class
predicted_class = model.predict(input_video)
predicted_label = np.argmax(predicted_class)
```

Step 3: Interpret the output:

```
label_map = {
  0: 'Boxing',
  1: 'HandWaving',
  2: 'Jogging',
  3: 'Running',
  4: 'Walking'
}

print("Predicted Action:", label_map[predicted_label])
```

📊 Results

- **Test Accuracy:** Achieved **75-80% accuracy** on the validation set.
- **Confusion Matrix:** Shows correct classifications for most classes but some overlap between similar actions like **Jogging** and **Running**.

Evaluation Command:

```
score = model.evaluate(X_val_new, y_val_new, batch_size=2)

print('Test accuracy:', score[1])
```

Sample Output:

```
Test accuracy: 0.78
```