Human Activity Detection System

What is an Activity Detection System?

This is system designed to identify and recognize specific actions or activities performed by humans.
These systems are commonly used in various fields including:

- Security
- Healthcare
- Sports
- Human - Computer Interaction

Libraries Used

```
# Import the required libraries.
import os #provides functions for interacting with the operating system.
import cv2 #allowing access to its functions for computer vision and image processing.
import pafy #a Python module used to download YouTube content and retrieve metadata.
import math #extends the list of mathematical functions.
import random #contains a number of random number generation-related functions.
import numpy as np #a Python library used for working with arrays.
import datetime as dt # imports all the content from the datetime module,
import tensorflow as tf # used to develop models for various tasks
from collections import deque #a Python data structure that efficiently adds and removes elements from both ends.
import matplotlib.pyplot as plt

from moviepy.editor import *
%matplotlib inline

from sklearn.model_selection import train_test_split

from tensorflow.keras.layers import *
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import plot_model
```

IMPORTING DATA

```
[7]  # Discards the output of the cell
     %%capture

     # Downloading the UCF50 Action dataset from the web
     !wget --no-check-certificate https://www.crcv.ucf.edu/data/UCF50.rar

     # Extract the dataset
     !unrar x UCF50.rar
```

Visualizing the dataset

```
# create and specify size of matplot figure
plt.figure(figsize = (20, 20))

# get names of all action categories
all_classes_names = os.listdir('UCF50')

# get 20 random categories
random_range = random.sample(range(len(all_classes_names)), 12)

# iterate through all the generated random values
for counter, random_index in enumerate(random_range, 1):

    # get the name of the random category
    selected_class_Name = all_classes_names[random_index]

    # get the list of all the video files present in selected_class_Name
    video_files_names_list = os.listdir(f'UCF50/{selected_class_Name}')

    # randomly select a video file from the list
    selected_video_file_name = random.choice(video_files_names_list)

    # video capture object to read the video file
    video_reader = cv2.VideoCapture(f'UCF50/{selected_class_Name}/{selected_video_file_name}')

    # read the first frame of the video file
    _, bgr_frame = video_reader.read()

    # release the video capture object
    video_reader.release()

    # convert the frame from BGR into RGB format
    rgb_frame = cv2.cvtColor(bgr_frame, cv2.COLOR_BGR2RGB)

    # write the class name on the video frame (USED TO WRITE ON THE VIDEO)
    cv2.putText(rgb_frame, selected_class_Name, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

    # display the frame
    plt.subplot(4, 3, counter)
    plt.imshow(rgb_frame)
    plt.axis('off')
```

Preprocess the data

```
# resizing the video frames in our dataset
img_height, img_width = 64, 64

# specify the number of frames of a video that will be fed to the model as one sequence
sequence_length = 20

# specifying the dataset directory
dataset_dir = "UCF50"

# Specifying the list of classes used for training the model
classes_list = ["PullUps", "BenchPress", "Punch", "PlayingGuitar", "PushUps"]
```

Function to Extract , Resize and Normalize the frames

```python
# function extracts the required frame from the video after resizing and normalizing it and then returns the a list of resized and normalized frames
def frame_extraction(video_path):
  frames_list = []

  # reading the video file using the VideoCapture object
  video_reader = cv2.VideoCapture(video_path)

  # counting the total number of frames in the video file
  video_frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))

  # calculating the interval after which frames will be added
  skip_frames_window = max(int(video_frames_count/sequence_length), 1)

  # iterate through the video frames
  for frame_counter in range(sequence_length):
    # set the current frame position of the video
    video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter * skip_frames_window)

    # read a frame from the video
    success, frame = video_reader.read()

    # check if the frame is not successfully read
    if not success:
      break

    # resize the frame
    resized_frame = cv2.resize(frame, (img_height, img_width))

    # normalize the resized frame by dividing it with 255 so that each pixel value then lies between 0 and 1
    normalized_frame = resized_frame / 255

    # append the normalized frame into the frames list
    frames_list.append(normalized_frame)

  # release the video capture object
  video_reader.release()

  # return the frames list
  return frames_list
```

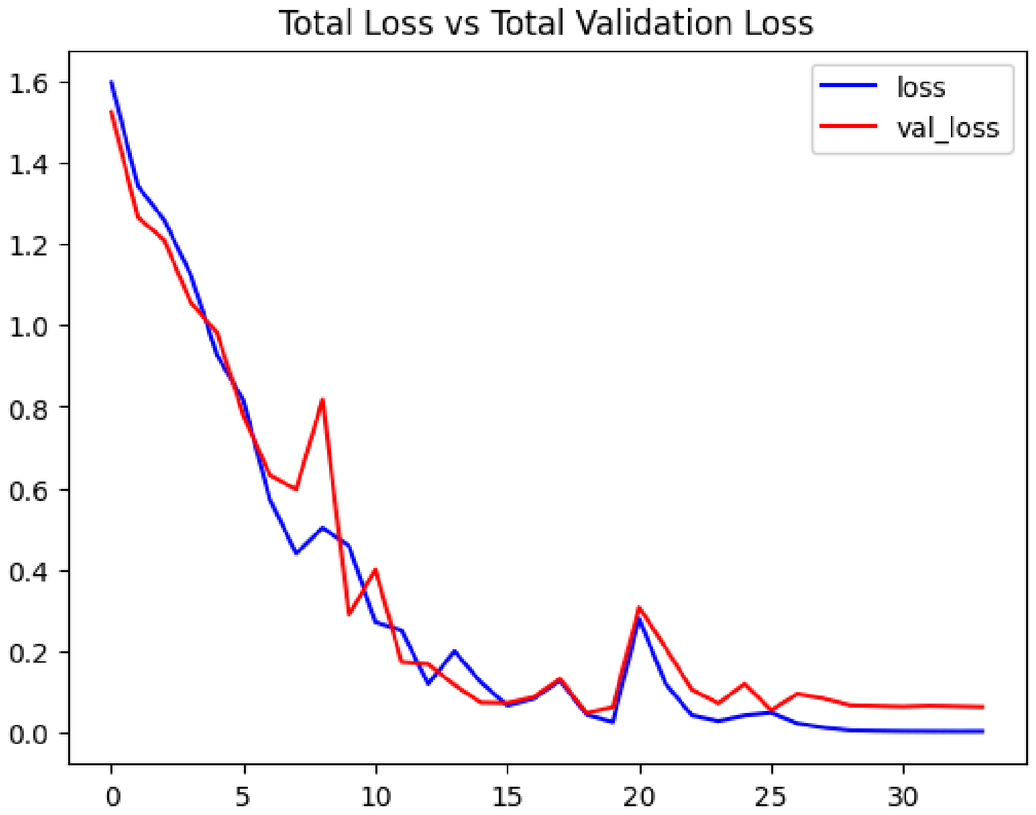
Manipulating DataSet

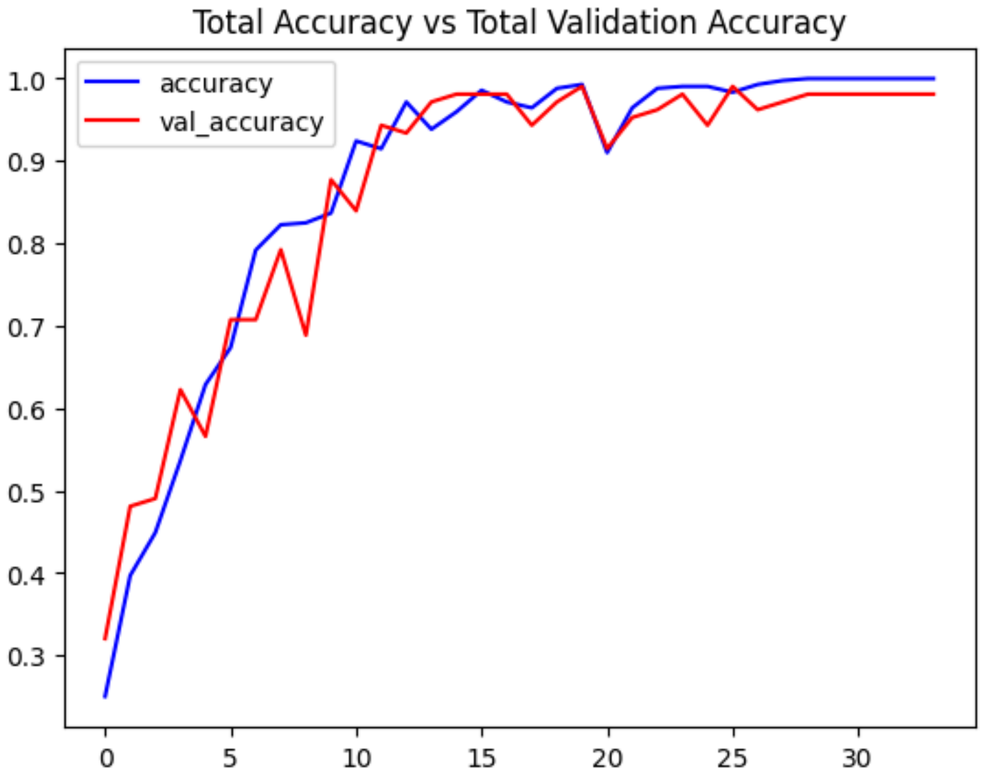
```python
# Function for dataset creation

def create_dataset():
  features = []
  labels = []
  video_files_paths = []

  # iterating through all the classes mentioned in the class list
  for class_index, class_name in enumerate(classes_list):

    # Display name of the class
    print(f'Extracting data from class: {class_name}')

    # get the list of video files present in the class
    files_list = os.listdir(os.path.join(dataset_dir, class_name))

    # iterate through all the files present in the files list
    for file_name in files_list:

      # get the video file path
      video_file_path = os.path.join(dataset_dir, class_name, file_name)

      # extract the frames from the video file
      frames = frame_extraction(video_file_path)

      # check if the extracted frames is equal to the sequence length i.e. 20
      # ignore videos having frames less than 20
      if len(frames) == sequence_length:

        # append the data
        features.append(frames)
        labels.append(class_index)
        video_files_paths.append(video_file_path)

  # convert the features and labels lists to numpy arrays
  features = np.asarray(features)
  labels = np.array(labels)

  # return the features, labels and video files paths
  return features, labels, video_files_paths
```

Implementing our Model

```
def create_LRCN_model():
  model = Sequential()

  model.add(TimeDistributed(Conv2D(16, (3, 3), padding = 'same', activation = 'relu'), input_shape = (sequence_length, img_height, img_width, 3)))
  model.add(TimeDistributed(MaxPooling2D((4, 4))))
  model.add(TimeDistributed(Dropout(0.25)))

  model.add(TimeDistributed(Conv2D(32, (3, 3), padding = 'same', activation = 'relu')))
  model.add(TimeDistributed(MaxPooling2D((4, 4))))
  model.add(TimeDistributed(Dropout(0.25)))

  model.add(TimeDistributed(Conv2D(64, (3, 3), padding = 'same', activation = 'relu')))
  model.add(TimeDistributed(MaxPooling2D((2, 2))))
  model.add(TimeDistributed(Dropout(0.25)))

  model.add(TimeDistributed(Conv2D(64, (3, 3), padding = 'same', activation = 'relu')))
  model.add(TimeDistributed(MaxPooling2D((2, 2))))
  # model.add(TimeDistributed(Dropout(0.25)))

  model.add(TimeDistributed(Flatten()))

  model.add(LSTM(32))

  model.add(Dense(len(classes_list), activation = 'softmax'))

  model.summary()

  return model
```

Model Accuracy and Loss Curves





Input

Output

Thank You!

Presented by

Vanshika Agarwal

Prakash Singh

Vineet Gupta

Saurabh Pandey