**Deep Learning-Based   
Sports Activity Recognition System for Video Analysis**

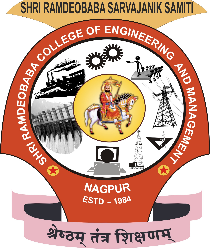
**MINIPROJECT**

Shri Ramdeobaba College of Engineering & Management

Nagpur-13

Department of Computer Application

Session: 2022-2023



Submission for

Course Name: Introduction to Deep Learning

Course Code: MCT646-1

Title: Deep Learning-Based Sports Activity Recognition System for Video Analysis

**Team Members: 33 – Manan Jain**

**53 – Rishi Sharma**

**Semester: MCA IV Semester**

**Shift: 1**

Under the Guidance of

### Prof. Aparna Gurjar

### Code:-

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

import cv2

import numpy as np

from keras.models import load\_model

import pickle

from collections import deque

import os

# Initialize global variables

selected\_video\_path = ""

selected\_image\_path = ""

video\_capture = None

model = None

lb = None

mean = None

Queue = None

writer = None

Width = None

Height = None

graph\_accuracy\_window = None

graph\_loss\_window = None

# Load the pre-trained model and other necessary objects

def load\_model\_and\_objects():

    global model, lb, mean

    try:

        model\_path = "E:/DeepLearning project/videoClassificationModel/videoClassificationModel"

        model = load\_model(model\_path)

        with open("E:/DeepLearning project/model/videoclassificationbinarizer.pickle", 'rb') as f:

            lb = pickle.load(f)

        mean = np.array([123.68, 116.779, 103.939][::1], dtype="float32")

    except Exception as e:

        print("Error loading model and objects:", e)

# Function to select a video file

def select\_video():

    global selected\_video\_path

    selected\_video\_path = filedialog.askopenfilename(filetypes=[("Video files", "\*.mp4;\*.avi")])

    if selected\_video\_path:

        process\_and\_display\_video()

# Function to select an image file

def select\_image():

    global selected\_image\_path

    selected\_image\_path = filedialog.askopenfilename(filetypes=[("Image files", "\*.jpg;\*.jpeg;\*.png")])

    if selected\_image\_path:

        process\_single\_frame()

# Function to process a single image frame

def process\_single\_frame():

    frame = cv2.imread(selected\_image\_path)

    if frame is None:

        print("Unable to read the image.")

        return

    frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

    frame = cv2.resize(frame, (244, 224)).astype("float32")

    frame -= mean

    preds = model.predict(np.expand\_dims(frame, axis=0))[0]

    label = lb.classes\_[np.argmax(preds)]

    accuracy = np.max(preds)

    show\_prediction(label, accuracy)

# Function to display the prediction result in a new window

def show\_prediction(label, accuracy):

    prediction\_window = tk.Toplevel()

    prediction\_window.title("Prediction Result")

    prediction\_label = tk.Label(prediction\_window, text="They are playing : " + label)

    prediction\_label.pack()

    accuracy\_label = tk.Label(prediction\_window, text="Accuracy: " + str(round(accuracy \* 100, 2)) + "%")

    accuracy\_label.pack()

# Function to process the video and display the output in the UI

# Function to process the video and display the output in the UI

def process\_and\_display\_video():

    global video\_capture, writer, Width, Height, Queue

    if selected\_video\_path == "" or model is None or lb is None or mean is None:

        print("Please make sure the model and video are selected.")

        return

    video\_capture = cv2.VideoCapture(selected\_video\_path)

    Queue = deque(maxlen=128)

    # Create a folder to store the frames

    frame\_folder = "E:/DeepLearning project/videoClassificationModel/Frames"

    if not os.path.exists(frame\_folder):

        try:

            os.makedirs(frame\_folder)

        except OSError as e:

            print("Error creating frame folder:", e)

            return

    frame\_count = 0

    correct\_predictions = 0

    total\_frames = 0

    while True:

        ret, frame = video\_capture.read()

        if not ret:

            break

        if Width is None or Height is None:

            (Height, Width) = frame.shape[:2]

        output = frame.copy()

        frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        frame = cv2.resize(frame, (244, 224)).astype("float32")

        frame = np.expand\_dims(frame, axis=0)

        frame -= mean

        preds = model.predict(frame)[0]

        correct\_predictions=np.max(preds)

        Queue.append(preds)

        results = np.array(Queue).mean(axis=0)

        i = np.argmax(results)

        label = lb.classes\_[i]

        text = "They are playing: {}".format(label)

        cv2.putText(output, text, (45, 60), cv2.FONT\_HERSHEY\_SIMPLEX, 1.25, (255, 0, 0), 5)

        accuracy = round(correct\_predictions \* 100, 2 )

        accuracy\_text = "Accuracy: {:.2f}%".format(accuracy)

        cv2.putText(output, accuracy\_text, (45, 90), cv2.FONT\_HERSHEY\_SIMPLEX, 1.0, (255, 0, 0), 3)

        # Save frame

        frame\_filename = os.path.join(frame\_folder, f"frame{frame\_count:04d}.png")

        try:

            cv2.imwrite(frame\_filename, output)

        except Exception as e:

            print(f"Error saving frame {frame\_count}: {e}")

        if writer is None:

            fourcc = cv2.VideoWriter\_fourcc(\*"XVID")

            output\_file = "E:/DeepLearning project/output\_video.avi"

            writer = cv2.VideoWriter(output\_file, fourcc, 30, (Width, Height), True)

        writer.write(output)

        cv2.imshow("In Progress", output)

        frame\_count += 1

        key = cv2.waitKey(1) & 0xFF

        if key == ord("q"):

            break

    Queue.clear()

    print("Finalizing")

    writer.release()

    video\_capture.release()

    cv2.destroyAllWindows()

# Function to show the accuracy graph

def show\_graph\_accuracy():

    global graph\_accuracy\_window

    if graph\_accuracy\_window is None:

        graph\_accuracy\_window = tk.Toplevel()

        graph\_accuracy\_window.title("Accuracy Graph")

        graph\_accuracy\_window.geometry("400x300")

    else:

        graph\_accuracy\_window.deiconify()

    # Load the graph image

    graph\_image\_path = r"E:\Developement\DeepLearning project\videoClassificationModel\model\_accuracy.png"

    graph\_image = Image.open(graph\_image\_path)

    graph\_image = graph\_image.resize((400, 300))

    # Convert the PIL image to a Tkinter-compatible format

    graph\_image\_tk = ImageTk.PhotoImage(graph\_image)

    # Display the graph image in the window

    graph\_label = tk.Label(graph\_accuracy\_window, image=graph\_image\_tk)

    graph\_label.image = graph\_image\_tk  # Keep a reference to prevent garbage collection

    graph\_label.pack()

# Function to show the loss graph

def show\_graph\_loss():

    global graph\_loss\_window

    if graph\_loss\_window is None:

        graph\_loss\_window = tk.Toplevel()

        graph\_loss\_window.title("Loss Graph")

        graph\_loss\_window.geometry("400x300")

    else:

        graph\_loss\_window.deiconify()

    # Load the graph image

    graph\_image\_path = r"E:\Developement\DeepLearning project\videoClassificationModel\model\_loss.png"  # Corrected path

    graph\_image = Image.open(graph\_image\_path)

    graph\_image = graph\_image.resize((400, 300))

    # Convert the PIL image to a Tkinter-compatible format

    graph\_image\_tk = ImageTk.PhotoImage(graph\_image)

    # Display the graph image in the window

    graph\_label = tk.Label(graph\_loss\_window, image=graph\_image\_tk)

    graph\_label.image = graph\_image\_tk  # Keep a reference to prevent garbage collection

    graph\_label.pack()

# Create the Tkinter window

window = tk.Tk()

window.title("Video Processing UI")

# Function to create a section with a title and a specified width

def create\_section(title):

    section\_frame = tk.Frame(window, bd=2, relief=tk.GROOVE)

    section\_frame.pack(side="top", padx=10, pady=10, fill="both", expand=True)

    section\_label = tk.Label(section\_frame, text=title, font=("Arial", 12, "bold"))

    section\_label.pack(pady=(10, 5))

    return section\_frame

# Add a background image

bg\_image = tk.PhotoImage(file="E:/Developement/DeepLearning project/deeplearningbg.png")

bg\_label = tk.Label(window, image=bg\_image)

bg\_label.place(relwidth=1, relheight=1)

# Create sections for image, video, and graphs

image\_section = create\_section("Image Processing")

video\_section = create\_section("Video Processing")

graph\_section = create\_section("Graphs")

# Add buttons for image processing

select\_image\_button = tk.Button(image\_section, text="Select Image", command=select\_image)

select\_image\_button.pack(pady=(0, 10))

process\_image\_button = tk.Button(image\_section, text="Process and Display Image", command=process\_single\_frame)

process\_image\_button.pack(pady=(0, 10))

# Add buttons for video processing

select\_video\_button = tk.Button(video\_section, text="Select Video", command=select\_video)

select\_video\_button.pack(pady=(0, 10))

process\_video\_button = tk.Button(video\_section, text="Process and Display Video", command=process\_and\_display\_video)

process\_video\_button.pack(pady=(0, 10))

# Add buttons for showing graphs

show\_Graph\_accuracy\_button = tk.Button(graph\_section, text="Show Accuracy Graph", command=show\_graph\_accuracy, bg="lightgreen")

show\_Graph\_accuracy\_button.pack(pady=(0, 10))

show\_Graph\_loss\_button = tk.Button(graph\_section, text="Show Loss Graph", command=show\_graph\_loss, bg="red")

show\_Graph\_loss\_button.pack(pady=(0, 10))

# Load the model and necessary objects when the window opens

load\_model\_and\_objects()

# Run the Tkinter event loop

window.mainloop()

### Output:-

### 