```python
import os
import pandas as pd
import numpy as np
import math
import matplotlib.colors as mcolors
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import traceback
# UMAP Approach:
from umap import UMAP

AVG_OPENFACE_FEATURE_COLUMNS_SELECTION = ["mean_AU01","mean_AU02","mean_AU04","mean_AU05","mean_AU06","mean_AU07","mean_AU09","mean_AU10","mean_AU11","mean_AU12","mean_AU14","mean_AU1

SEQUENTIAL_OPENFACE_FEATURE_COLUMNS_SELECTION = ["AU01","AU02","AU04","AU05","AU06","AU07","AU09","AU10","AU11","AU12","AU14","AU15","AU17","AU20","AU23","AU24","AU25","AU26","AU28",'

SEQUENTIAL_MEDIAPIPE_FEATURE_COLUMNS_SELECTION = [468, 473, 282, 52, 4, 0, 16, 40, 90, 270, 320, 199]

class CautDataloaderRegular:
    # get dataframes for train, val, and test:
    @staticmethod
    def get_TrainValTest_dfs(csv_path):
        print("\nSelected csv_path:", csv_path)
        train_df = pd.read_csv(os.path.join(csv_path, "train_DARE.csv"))
        val_df = pd.read_csv(os.path.join(csv_path, "val_DARE.csv"))
        pre_test_df = pd.read_csv(os.path.join(csv_path, "test_DARE.csv"))

        val_plus_test = [val_df, pre_test_df]
        test_df = pd.concat(val_plus_test)
        test_df.reset_index(drop=True, inplace=True)

        return train_df, test_df


    # get X_train, y_train, X_test, y_test
    @staticmethod
    def get_TrainTest_meta_csv(csv_path):
        # get train and test dataframes:
        train_df, test_df = CautDataloaderRegular.get_TrainValTest_dfs(csv_path)

        return train_df, test_df


    # duplicate frames if FPS is lower than expected. Add zeros to the rest, if needed.
    @staticmethod
    def standardize_FPS(data, frame_cap):
        # if FPS is lower, duplicate every frame to increase (generate slow video as workaround):
        if data.shape[0] < frame_cap:
            repeat_for = int(math.ceil(frame_cap / data.shape[0]))
            data = np.repeat(data, repeat_for, axis=0)[:frame_cap]
            # add additional edge case carry out:
            if len(data) < frame_cap:
                extra_needed = frame_cap - len(data)
                extra_array = np.zeros((extra_needed, data.shape[1]), dtype=float)
                data = np.concatenate((data, extra_array))
        return data[:frame_cap]



    # retrieve X_data and y_data from MediaPipe.
    # sequential column selection: ["AU01","AU02","AU04","AU05","AU06","AU07","AU09","AU10","AU11","AU12","AU14","AU15","AU17","AU20","AU23","AU24","AU25","AU26","AU28","AU43","anger",
    # average column selection: ["mean_AU01","mean_AU02","mean_AU04","mean_AU05","mean_AU06","mean_AU07","mean_AU09","mean_AU10","mean_AU11","mean_AU12","mean_AU14","mean_AU15","mean_A
    @staticmethod
    def get_Xy_data_OpenFace(data_dir,
                             meta_df,
                             required_FPS,
                             input_length_in_seconds,
                             class_to_num_dict,
                             # coord_selection,
                             approach_type,
                             verbose):
        X_data = []
        y_data = []
        # frame amount * input seconds, e.g. 29*3=>87 => expected input length
        frame_cap = required_FPS*input_length_in_seconds
        counter, total = 0, len(meta_df)
        # if we chose average approach, then let's locate Final.csv (each row represents single video):
        if approach_type == "average":
            # get array with averaged video impressions:
            avg_df = pd.read_csv(os.path.join(data_dir, "Final.csv"))
            for filename in meta_df["video_name"]:
                try:
                    current_label_name = filename.split("_")[1]
                    current_label_num  = class_to_num_dict[current_label_name]
                    # find the row with that filename:
                    current_video_name = filename.replace(".mp4", "")
                    current_row_data = avg_df.loc[avg_df['VideoName'] == current_video_name]
                    if len(current_row_data) <= 0:
                        print(f"Video failed to be processed by OpenFace. Videoname={current_video_name}.")
                        continue
                    current_data = current_row_data[AVG_OPENFACE_FEATURE_COLUMNS_SELECTION].iloc[0].values
                    # append to X and y data:
                    # no need to trim sequences, since we deal with video averaged impressions
                    X_data.append(current_data)
                    y_data.append(current_label_num)
                    # keep track of how many things we gathered:
                    counter+=1
                    if verbose:
                        if (counter%100==0):
                            print(f"Processed {counter} / {total}")
                            print("  - Sample shape & label:")
                            print(f"    - X_data: {current_data.shape}")
                            print(f"    - y_data: {current_label_num}")
                except:  # if we get error in other loops (Sequential OpenFace or MediaPipe), then add this error exception there too)
                    print("##########################################################################")
                    print(">>> ERROR:")
                    print(f"Failed with retrieving data for videoname={current_video_name}. Please check the error below...")
                    print(traceback.format_exc())
                    print("##########################################################################\n")

        # if it is sequential, we need to read from list of .csv files, each representing individual video.
        elif approach_type == "sequential":
            for filename in meta_df["video_name"]:
                # setup label name and num:
                current_label_name = filename.split("_")[1]
                current_label_num  = class_to_num_dict[current_label_name]
                # get path to data:
                path = os.path.join(data_dir, f"{filename.replace('.mp4', '')}.csv")
                # if we have such path, then we read it, get features of interest,
                # and reshape into (frame, features*xyz)
                # print("path:", path)
                # print("os.path.exists(path):", os.path.exists(path))
                if(os.path.exists(path)):
                    arr = pd.read_csv(path)
                    current_data = arr[SEQUENTIAL_OPENFACE_FEATURE_COLUMNS_SELECTION].values

                    # standardize data to same FPS:
                    current_data_processed = CautDataloaderRegular.standardize_FPS(data=current_data,
                                                                                   frame_cap=frame_cap)

                    # record data:
                    X_data.append(current_data_processed)
                    y_data.append(current_label_num)
                    # keep track of how much we have processed...
                    counter+=1
                    if verbose:
                        if (counter%100==0):
                            print(f"Processed {counter} / {total}")
                            print("  - Sample shape & label:")
                            print(f"    - X_data: {current_data_processed.shape}")
                            print(f"    - y_data: {current_label_num}")
        else:
            print(f">>> ERROR: No such approach is implemented ({approach_type})")
            return np.array([]), np.array([])

        print("Casting collected data to .npy array type...")
        X_data = np.array(X_data)
        y_data = np.array(y_data)
        print("Data is collected. Returning X and y data.")
        print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n\n")
        return X_data, y_data



    # retrieve X_data and y_data from MediaPipe.
    @staticmethod
    def get_Xy_data_mediaPipe(data_dir,
                              meta_df,
                              required_FPS,
                              input_length_in_seconds,
                              class_to_num_dict,
                              # coord_selection,
                              verbose):
        X_data = []
        y_data = []
        # frame amount * input seconds, e.g. 29*3=>87 => expected input length
        frame_cap = required_FPS*input_length_in_seconds
        counter, total = 0, len(meta_df)

        for filename in meta_df["video_name"]:
            # setup label name and num:
            current_label_name = filename.split("_")[1]
            current_label_num  = class_to_num_dict[current_label_name]
            # get path to data:
            path = os.path.join(data_dir, f"{filename.replace('.mp4', '')}_MP_coord.npy")
            # if we have such path, then we read it, get features of interest,
            # and reshape into (frame, features*xyz)
            if(os.path.exists(path)):
                arr = np.load(path)
                trun_arr = arr[:,SEQUENTIAL_MEDIAPIPE_FEATURE_COLUMNS_SELECTION,:]
                dim_1, dim_2, dim_3 = trun_arr.shape[0], trun_arr.shape[1], trun_arr.shape[2]
                current_data = trun_arr.reshape((dim_1, dim_2*dim_3))

                # standardize data to same FPS:
                current_data_processed = CautDataloaderRegular.standardize_FPS(data=current_data,
                                                                               frame_cap=frame_cap)

                # record data:
                X_data.append(current_data_processed)
                y_data.append(current_label_num)
                # keep track of how much we have processed...
                counter+=1
                if verbose:
                    if (counter%100==0):
                        print(f"Processed {counter} / {total}")
                        print("  - Sample shape & label:")
                        print(f"    - X_data: {current_data_processed.shape}")
                        print(f"    - y_data: {current_label_num}")

        X_data = np.array(X_data)
        y_data = np.array(y_data)

        return X_data, y_data



    # retrieve X_data and y_data from MediaPipe.
    @staticmethod
    def get_Xy_data_audioFeatures(data_dir,
                                  meta_df,
                                  input_length_in_seconds,
                                  feature_type,
                                  class_to_num_dict,
                                  # coord_selection,
                                  verbose):
        X_data = []
        y_data = []
        # frame amount * input seconds, e.g. 29*3=>87 => expected input length
        counter, total = 0, len(meta_df)

        for filename in meta_df["video_name"]:
            # setup label name and num:
            current_label_name = filename.split("_")[1]
            current_label_num  = class_to_num_dict[current_label_name]
            # get path to data:
            path = os.path.join(data_dir, f"{filename.replace('.mp4', '')}_{feature_type}.npy")
            # if we have such path, then we read it, get features of interest,
            # and reshape into (frame, features*xyz)
            if(os.path.exists(path)):
                arr = np.load(path)
                trun_arr = arr[:input_length_in_seconds*1000]  # audio is trimmed in milliseconds
                current_data = np.transpose(trun_arr, (1,0))

                # record data:
                X_data.append(current_data)
                y_data.append(current_label_num)
                # keep track of how much we have processed...
                counter+=1
                if verbose:
                    if (counter%100==0):
                        print(f"Processed {counter} / {total}")
                        print("  - Audio sample shape & label:")
                        print(f"    - X_data: {current_data.shape}")
                        print(f"    - y_data: {current_label_num}")

        X_data = np.array(X_data)
        y_data = np.array(y_data)

        return X_data, y_data



    # retrieve visual dataset (OpenFace or MediaPipe):
    @staticmethod
    def get_X_y_TrainTest_Visual(csv_path,
                                 data_dir,
                                 data_mode,
                                 # coord_selection,
                                 approach_type=None,  # average or frame-based
                                 required_FPS = 30,
                                 input_length_in_seconds = 3,
                                 class_to_num_dict = {"truth": 0, "lie": 1},
                                 verbose = True):

        # meta data:
        train_df_meta, test_df_meta = CautDataloaderRegular.get_TrainTest_meta_csv(csv_path)

        # get actual data
        data_mode = data_mode.lower()
        ##################
        # OPENFACE MODE: #
        ##################
        if data_mode == "openface":
            # train:
            X_train, y_train = CautDataloaderRegular.get_Xy_data_OpenFace(data_dir=data_dir,
                                                                          meta_df=train_df_meta,
                                                                          required_FPS=required_FPS,
                                                                          input_length_in_seconds=input_length_in_seconds,
                                                                          class_to_num_dict=class_to_num_dict,
                                                                          # coord_selection=coord_selection,
                                                                          approach_type=approach_type,
                                                                          verbose=verbose)
            # test:
            X_test, y_test = CautDataloaderRegular.get_Xy_data_OpenFace(data_dir=data_dir,
                                                                        meta_df=test_df_meta,
                                                                        required_FPS=required_FPS,
                                                                        input_length_in_seconds=input_length_in_seconds,
                                                                        class_to_num_dict=class_to_num_dict,
                                                                        # coord_selection=coord_selection,
                                                                        approach_type=approach_type,
                                                                        verbose=verbose)
            return X_train, y_train, X_test, y_test
        ###################
        # MEDIAPIPE MODE: #
        ###################
        elif data_mode == "mediapipe":
            # train:
            X_train, y_train = CautDataloaderRegular.get_Xy_data_mediaPipe(data_dir=data_dir,
                                                                           meta_df=train_df_meta,
                                                                           required_FPS=required_FPS,
                                                                           input_length_in_seconds=input_length_in_seconds,
                                                                           class_to_num_dict=class_to_num_dict,
                                                                           # coord_selection=coord_selection,
                                                                           verbose=verbose)
            # test:
            X_test, y_test = CautDataloaderRegular.get_Xy_data_mediaPipe(data_dir=data_dir,
                                                                         meta_df=test_df_meta,
                                                                         required_FPS=required_FPS,
                                                                         input_length_in_seconds=input_length_in_seconds,
                                                                         class_to_num_dict=class_to_num_dict,
                                                                         # coord_selection=coord_selection,
                                                                         verbose=verbose)
            # check on results:
            if verbose:
                print("----------------------------")
                print("Gathered data shapes:")
                print("X_train.shape:", X_train.shape)
                print("y_train.shape:", y_train.shape)
                print("X_test.shape:", X_test.shape)
                print("y_test.shape:", y_test.shape)
            return X_train, y_train, X_test, y_test
        ###############################
        # NO OTHER MODE WAS ADDED YET #
        ###############################
        else:
            print(f">>> ERROR: No such supported data_mode = {data_mode}")
            return None, None, None, None

        # if we reached here, it's odd:
        print(">>> Unknown behavior...")
        return None, None, None, None



    # retrieve audio dataset:
    @staticmethod
    def get_X_y_TrainTest_Audio(csv_path,
                                data_dir,
                                # coord_selection,
                                feature_type="MFCC",  # MFCC, RMS, Chroma
                                input_length_in_seconds = 3,
                                class_to_num_dict = {"truth": 0, "lie": 1},
                                verbose = True):

        data_dir = os.path.join(data_dir, f"{feature_type}_audio_features")
        print(f"data_dir updated to: {data_dir}")

        # meta data:
        train_df_meta, test_df_meta = CautDataloaderRegular.get_TrainTest_meta_csv(csv_path)

        # get actual data
        ##################
        # Audio MODE: #
        ##################
        # train:
        X_train, y_train = CautDataloaderRegular.get_Xy_data_audioFeatures(data_dir=data_dir,
                                                                           meta_df=train_df_meta,
                                                                           input_length_in_seconds=input_length_in_seconds,
                                                                           feature_type=feature_type,
                                                                           class_to_num_dict=class_to_num_dict,
                                                                           verbose=verbose)
        # test:
        X_test, y_test = CautDataloaderRegular.get_Xy_data_audioFeatures(data_dir=data_dir,
                                                                         meta_df=test_df_meta,
                                                                         input_length_in_seconds=input_length_in_seconds,
                                                                         feature_type=feature_type,
                                                                         class_to_num_dict=class_to_num_dict,
                                                                         verbose=verbose)
        # check on results:
        if verbose:
            print("----------------------------")
            print("Gathered data shapes:")
            print("X_train.shape:", X_train.shape)
            print("y_train.shape:", y_train.shape)
            print("X_test.shape:", X_test.shape)
            print("y_test.shape:", y_test.shape)
        return X_train, y_train, X_test, y_test

        ##############################
        # NO OTHER MODE WAS ADDED YET #
        ##############################

        # if we reached here, it's odd:
        # print(">>> Unknown behavior...")
        # return None, None, None, None



    # retrieve audio dataset:
    @staticmethod
    def get_X_y_TrainTest_Fused(csv_path,
                                visual_data_dir,
                                visual_data_mode,
                                audio_data_dir,
                                # coord_selection,
                                fusion_mode,
                                visual_approach_type=None,  # average or frame-based
                                required_FPS = 30,
                                input_length_in_seconds = 3,
                                audio_feature_type="MFCC",  # MFCC, RMS, Chroma
                                class_to_num_dict = {"truth": 0, "lie": 1},
                                verbose = True):

        # visual_data_dir => for visual data path.
        audio_data_dir = os.path.join(audio_data_dir, f"{audio_feature_type}_audio_features")  # for audio data path.
        print(f"audio_data_dir updated to: {audio_data_dir}")

        # meta data:
        train_df_meta, test_df_meta = CautDataloaderRegular.get_TrainTest_meta_csv(csv_path)

        # get actual data
        #################
        # Audio MODE: #
        #################
        # train:
        X_train, y_train = CautDataloaderRegular.get_Xy_data_fusedFeatures(meta_df=train_df_meta,
                                                                           visual_data_dir=visual_data_dir,
                                                                           visual_data_mode=visual_data_mode,
                                                                           audio_data_dir=audio_data_dir,
                                                                           # coord_selection,
                                                                           fusion_mode=fusion_mode,  # or "+"
                                                                           visual_approach_type=visual_approach_type,  # average or frame-based
                                                                           required_FPS=required_FPS,
                                                                           input_length_in_seconds=input_length_in_seconds,
                                                                           audio_feature_type=audio_feature_type,  # MFCC, RMS, Chroma
                                                                           class_to_num_dict=class_to_num_dict,
                                                                           verbose=verbose)
        # test:
        X_test, y_test = CautDataloaderRegular.get_Xy_data_fusedFeatures(meta_df=test_df_meta,
                                                                         visual_data_dir=visual_data_dir,
                                                                         visual_data_mode=visual_data_mode,
                                                                         audio_data_dir=audio_data_dir,
                                                                         # coord_selection,
                                                                         fusion_mode=fusion_mode,  # or "+"
                                                                         visual_approach_type=visual_approach_type,  # average or frame-based
                                                                         required_FPS=required_FPS,
                                                                         input_length_in_seconds=input_length_in_seconds,
                                                                         audio_feature_type=audio_feature_type,  # MFCC, RMS, Chroma
                                                                         class_to_num_dict=class_to_num_dict,
                                                                         verbose=verbose)
        # check on results:
        if verbose:
            print("----------------------------")
            print("Gathered data shapes:")
            print("X_train.shape:", X_train.shape)
            print("y_train.shape:", y_train.shape)
            print("X_test.shape:", X_test.shape)
            print("y_test.shape:", y_test.shape)
        return X_train, y_train, X_test, y_test

    ########################################################################################################################
    ############################################# FUSION FUNCTIONS #########################################################
    ########################################################################################################################
    @staticmethod
    def get_audio_sample(data_dir, filename, feature_type, seconds_limit):
        audio_path = os.path.join(data_dir, f"{filename.replace('.mp4', '')}_{feature_type}.npy")
        if(os.path.exists(audio_path)):
            arr = np.load(audio_path)
            trun_arr = arr[:seconds_limit*1000]  # audio is trimmed in milliseconds
            audio_data = np.transpose(trun_arr, (1,0))
            return audio_data
        return None


    @staticmethod
    def get_fused_features(visual_feature, audio_feature, visual_approach_type, fusion_mode, frame_cap):

        # print("Attempting to fuse:")
        # print(f"  - visual feature: {visual_feature.shape}")
        # print(f"  - audio feature: {audio_feature.shape}")

        if len(visual_feature.shape) > 1:
         visual_feature_dim = visual_feature.shape[1]
        else:
         visual_feature_dim = 1

        if len(audio_feature.shape) > 1:
         audio_feature_dim = audio_feature.shape[1]
        else:
         audio_feature_dim = 1
```

```python
        # trim to visual feature shape:
        audio_feature = audio_feature[:frame_cap]  # think of a better way in the meantime.

        fused_feature = None
        if visual_approach_type == "average":
            if fusion_mode == "x":
                fused_feature = np.multiply(visual_feature, audio_feature)
            else:  # the fusion_mode will be "+"
                fused_feature = np.concatenate((visual_feature, np.mean(audio_feature, axis=0)), axis=0)
        elif visual_approach_type == "sequential":  # means that visual_data_mode == "sequential":
            if fusion_mode == "x":
                if visual_feature_dim > audio_feature_dim:
                    umap_3d = UMAP(n_components=audio_feature_dim, init='random', random_state=0)
                    visual_feature = umap_3d.fit_transform(visual_feature)
                else:
                    umap_3d = UMAP(n_components=visual_feature_dim, init='random', random_state=0)
                    audio_feature = umap_3d.fit_transform(audio_feature)
                fused_feature = np.multiply(visual_feature, audio_feature)
            else:  # the fusion_mode will be "+"
                fused_feature = np.concatenate((visual_feature, audio_feature), axis = 1)
        else:
            print(f">>> ERROR: No such supported visual_data_mode = {visual_approach_type}")

        # if not (fused_feature is None):
        #     print(f"\n>>> Fused feature result shape: {fused_feature.shape}")

        return fused_feature




    @staticmethod
    def get_Xy_data_fusedFeatures(meta_df,
                                  visual_data_dir,
                                  visual_data_mode,
                                  audio_data_dir,
                                  # coord_selection,
                                  fusion_mode,  # or "+"
                                  visual_approach_type,  # average or frame-based
                                  required_FPS,
                                  input_length_in_seconds,
                                  audio_feature_type,  # MFCC, RMS, Chroma
                                  class_to_num_dict,
                                  verbose):

        X_data = []
        y_data = []

        frame_cap = required_FPS*input_length_in_seconds

        counter, total = 0, len(meta_df)

        visual_data_mode = visual_data_mode.lower()

        if visual_data_mode == "openface":
            # do OpenFace + Audio fusion here
            # if we chose average approach, then let's locate Final.csv (each row represents single video):
            if visual_approach_type == "average":
                # get array with averaged video impressions:
                avg_df = pd.read_csv(os.path.join(visual_data_dir, "Final.csv"))
                for filename in meta_df["video_name"]:
                    try:
                        # get label and associated num:
                        current_label_name = filename.split("_")[1]
                        current_label_num  = class_to_num_dict[current_label_name]

                        # find the row with that filename:
                        current_video_name = filename.replace(".mp4", "")
                        current_row_data = avg_df.loc[avg_df['VideoName'] == current_video_name]
                        if len(current_row_data) <= 0:
                            print(f"Video failed to be processed by OpenFace. Videoname={current_video_name}. Skipping...")
                            continue

                        # GET VISUAL DATA:
                        current_data = current_row_data[AVG_OPENFACE_FEATURE_COLUMNS_SELECTION].iloc[0].values

                        # GET AUDIO DATA:
                        current_audio_data = CautDataloaderRegular.get_audio_sample(data_dir=audio_data_dir,
                                                                                    filename=filename,
                                                                                    feature_type=audio_feature_type,
                                                                                    seconds_limit=input_length_in_seconds)
                        # GET FUSED DATA:
                        fused_data = CautDataloaderRegular.get_fused_features(visual_feature=current_data,
                                                                              audio_feature=current_audio_data,
                                                                              visual_approach_type=visual_approach_type,
                                                                              fusion_mode=fusion_mode,
                                                                              frame_cap=1)

                        if not (fused_data is None):
                            # append to X and y data:
                            # no need to trim sequences, since we deal with video averaged impressions
                            X_data.append(fused_data)
                            y_data.append(current_label_num)
                            # keep track of how many things we gathered:
                            counter+=1
                            if verbose:
                                if (counter%100==0):
                                    print(f"Processed {counter} / {total}")
                                    print("  - Sample shape & label:")
                                    print(f"    - X_data: {fused_data.shape}")
                                    print(f"    - y_data: {current_label_num}")
                    except:  # if we get error in other loops (Sequential OpenFace or MediaPipe), then add this error exception there too)
                        print("##########################################################################")
                        print(">>> ERROR:")
                        print(f"Failed with retrieving data for videoname={current_video_name}. Please check the error below...")
                        print(traceback.format_exc())
                        print("##########################################################################\n")
            # if it is sequential, we need to read from list of .csv files, each representing individual video.
            elif visual_approach_type == "sequential":  # "sequential":
                for filename in meta_df["video_name"]:
                    try:
                        # setup label name and num:
                        current_label_name = filename.split("_")[1]
                        current_label_num  = class_to_num_dict[current_label_name]
                        # get path to data:
                        path = os.path.join(visual_data_dir, f"{filename.replace('.mp4', '')}.csv")
                        # if we have such path, then we read it, get features of interest,
                        # and reshape into (frame, features*xyz)
                        # print("path:", path)
                        # print("os.path.exists(path):", os.path.exists(path))
                        if(os.path.exists(path)):
                            arr = pd.read_csv(path)
                            current_data = arr[SEQUENTIAL_OPENFACE_FEATURE_COLUMNS_SELECTION].values

                            # GET VISUAL DATA:
                            # standardize data to same FPS:
                            current_data_processed = CautDataloaderRegular.standardize_FPS(data=current_data,
                                                                                           frame_cap=frame_cap)

                            # GET AUDIO DATA:
                            current_audio_data = CautDataloaderRegular.get_audio_sample(data_dir=audio_data_dir,
                                                                                        filename=filename,
                                                                                        feature_type=audio_feature_type,
                                                                                        seconds_limit=input_length_in_seconds)

                            # GET FUSED DATA:
                            fused_data = CautDataloaderRegular.get_fused_features(visual_feature=current_data_processed,
                                                                                  audio_feature=current_audio_data,
                                                                                  visual_approach_type=visual_approach_type,
                                                                                  fusion_mode=fusion_mode,
                                                                                  frame_cap=frame_cap)

                            if not (fused_data is None):
                                # record data:
                                X_data.append(fused_data)
                                y_data.append(current_label_num)
                                # keep track of how much we have processed...
                                counter+=1
                                if verbose:
                                    if (counter%100==0):
                                        print(f"Processed {counter} / {total}")
                                        print("  - Sample shape & label:")
                                        print(f"    - X_data: {fused_data.shape}")
                                        print(f"    - y_data: {current_label_num}")
                    except:
                        print("##########################################################################")
                        print(">>> ERROR:")
                        print(f"Failed with retrieving data for videoname={filename}. Please check the error below...")
                        print(traceback.format_exc())
                        print("##########################################################################\n")

            else:
                print(f">>> ERROR: No such supported visual_approach_type = {visual_approach_type}")

        elif visual_data_mode == "mediapipe":  # otherwise, take care of MediaPipe
            # do MediaPipe + Audio fusion here
            for filename in meta_df["video_name"]:
                try:
                    # setup label name and num:
                    current_label_name = filename.split("_")[1]
                    current_label_num  = class_to_num_dict[current_label_name]
                    # get path to data:
                    path = os.path.join(visual_data_dir, f"{filename.replace('.mp4', '')}_MP_coord.npy")
                    # if we have such path, then we read it, get features of interest,
                    # and reshape into (frame, features*xyz)
                    if(os.path.exists(path)):
                        arr = np.load(path)
                        trun_arr = arr[:,SEQUENTIAL_MEDIAPIPE_FEATURE_COLUMNS_SELECTION,:]
                        dim_1, dim_2, dim_3 = trun_arr.shape[0], trun_arr.shape[1], trun_arr.shape[2]
                        current_data = trun_arr.reshape((dim_1, dim_2*dim_3))

                        # GET VISUAL DATA:
                        # standardize data to same FPS:
                        current_data_processed = CautDataloaderRegular.standardize_FPS(data=current_data,
                                                                                       frame_cap=frame_cap)

                        # GET AUDIO DATA:
                        current_audio_data = CautDataloaderRegular.get_audio_sample(data_dir=audio_data_dir,
                                                                                    filename=filename,
                                                                                    feature_type=audio_feature_type,
                                                                                    seconds_limit=input_length_in_seconds)

                        # GET FUSED DATA:
                        fused_data = CautDataloaderRegular.get_fused_features(visual_feature=current_data_processed,
                                                                              audio_feature=current_audio_data,
                                                                              visual_approach_type=visual_approach_type,
                                                                              fusion_mode=fusion_mode,
                                                                              frame_cap=frame_cap)

                        if not (fused_data is None):
                            # record data:
                            X_data.append(fused_data)
                            y_data.append(current_label_num)
                            # keep track of how much we have processed...
                            counter+=1
                            if verbose:
                                if (counter%100==0):
                                    print(f"Processed {counter} / {total}")
                                    print("  - Sample shape & label:")
                                    print(f"    - X_data: {fused_data.shape}")
                                    print(f"    - y_data: {current_label_num}")

                except:
                        print("##########################################################################")
                        print(">>> ERROR:")
                        print(f"Failed with retrieving data for videoname={filename}. Please check the error below...")
                        print(traceback.format_exc())
                        print("##########################################################################\n")

        else:
            print(f">>> ERROR: No such supported data_mode = {data_mode}")

        # return results:
        X_data = np.array(X_data)
        y_data = np.array(y_data)

        return X_data, y_data
    #########################################################################################################################
    ########################################## END OF FUSION FUNCTIONS ######################################################
    #########################################################################################################################



    @staticmethod
    def get_positive_negative_rates(confusion_matrix):
        #calculate false positive, false negative, true positive and true nagative
        TN = confusion_matrix[0, 0]
        FP = confusion_matrix[0, 1]
        FN = confusion_matrix[1, 0]
        TP = confusion_matrix[1, 1]
        # Sensitivity, hit rate, recall, or true positive rate
        TPR = TP/(TP+FN)
        # Specificity or true negative rate
        TNR = TN/(TN+FP)
        # Precision or positive predictive value
        PPV = TP/(TP+FP)
        # Negative predictive value
        NPV = TN/(TN+FN)
        # Fall out or false positive rate
        FPR = FP/(FP+TN)
        # False negative rate
        FNR = FN/(TP+FN)
        # False discovery rate
        FDR = FP/(TP+FP)

        print("\nMetrics Rates:")
        print("     - True Positive            :", TP)
        print("     - False Positive           :", FP)
        print("     - True Negative            :", TN)
        print("     - False Negative           :", FN)
        print("     - True Positive Rate       : ",TPR)
        print("     - True Negative Rate       : ",TNR)
        print("     - Positive Predictive Value: ",PPV)
        print("     - Negative predictive value: ",NPV)
        print("     - False Positive Rate      : ",FPR)
        print("     - False Negative Rate      : ",FNR)
        print("     - False Discovery Rate     : ",FDR)


    @staticmethod
    def plot_confusion_matrix(y_test, y_pred):

        # get confusion matrix:
        conf_matrix = confusion_matrix(y_test, y_pred)

        cmap = mcolors.LinearSegmentedColormap.from_list('custom', ['#AFE1AF', '#1c4a60'])
        classes = np.array([False,True])
        fig, ax = plt.subplots()
        im = ax.imshow(conf_matrix, interpolation='nearest', cmap=cmap)
        ax.figure.colorbar(im, ax=ax)
        ax.set(xticks=np.arange(conf_matrix.shape[1]),
               yticks=np.arange(conf_matrix.shape[0]),
               xticklabels=classes, yticklabels=classes,
               ylabel='True label',
               xlabel='Predicted label')
        thresh = (conf_matrix.max() + conf_matrix.min()) / 2.0
        for i in range(conf_matrix.shape[0]):
            for j in range(conf_matrix.shape[1]):
                ax.text(j, i, format(conf_matrix[i, j], 'd'),
                        ha="center", va="center",
                        color="white" if conf_matrix[i, j] > thresh else "black")
        fig.tight_layout()
        plt.show()

        print("---------------------------------------------------------------")
        CautDataloaderRegular.get_positive_negative_rates(conf_matrix)
```