Escuela Universitaria Politécnica - La Almunia
Centro adscrito
Universidad Zaragoza

**ESCUELA UNIVERSITARIA POLITÉCNICA**
**DE LA ALMUNIA DE DOÑA GODINA (ZARAGOZA)**

# ANEXOS

## Reconocimiento y clasificación de Emociones Empleando Redes Neuronales

## Emotion Recognition and Classification Using Neural Networks

424.22.28


Autor: Ilmira Iembergenova

Director: Dr. David Asiain Ansorena


Fecha: 09 2023

# ÍNDICE DE CONTENIDO

# 1. PROCESAMIENTO Y EXTRACCIÓN DE EMOCIONES DE IMÁGENES (MTCNN)

```
import pyxdf
from fer import FER
import cv2
import os
import pandas as pd

Dict = {'angry': [],
        'disgust': [],
        'fear': [],
        'happy': [],
        'sad': [],
        'surprise': [],
        'neutral': []
        }
index_time_stamp = 0

# path_name = r'D:\Imagenes\img\01_31_09_53_31\window'
path_name = r'D:\Imagenes\img\01_31_13_00_11\window'
# path_name = r'D:\Imagenes\img\02_01_12_04_06\window'

# dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0020\ses-
martes\eeg\sub-P0020_ses-martes_task-Default_run-
001_eeg.xdf")
dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0023\ses-
martes\eeg\sub-P0023_ses-martes_task-Default_run-
001_eeg.xdf")
# dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0026\ses-
miercoles\eeg\sub-P0026_ses-miercoles_task-Default_run-
001_eeg.xdf")

for stream in dataf:
    for i in range(len(dataf)):
        r = dataf[i]['info']['name']
        if r == ['cam']:
            cam_series = (dataf[i]['time_series'])
            cam_stamp = (dataf[i]['time_stamps'])
        else:
            pass
image_modified =
r'C:\Users\Mirus\PycharmProjects\Thesis\S3_processed_images'
```

```
for arr in cam_series[:19043]:
    i = arr[0]
    filename = path_name + '\\' + 'i' + str(i) + '.jpg'
    img = cv2.imread(filename)
    if img is None or img.size == 0:
        Dict['angry'].append(-1)
        Dict['disgust'].append(-1)
        Dict['fear'].append(-1)
        Dict['happy'].append(-1)
        Dict['sad'].append(-1)
        Dict['surprise'].append(-1)
        Dict['neutral'].append(-1)
        index_time_stamp += 1
        continue
    else:
        fer_emo_detector = FER(mtcnn=True)
        class_emo = fer_emo_detector.detect_emotions(img)
        predom_emo, score_emo =
fer_emo_detector.top_emotion(img)
        if not class_emo:
            Dict['angry'].append(-1)
            Dict['disgust'].append(-1)
            Dict['fear'].append(-1)
            Dict['happy'].append(-1)
            Dict['sad'].append(-1)
            Dict['surprise'].append(-1)
            Dict['neutral'].append(-1)
            index_time_stamp += 1
        elif len(class_emo[0]['box']) == 4:

Dict['angry'].append(class_emo[0]['emotions']['angry'])

Dict['disgust'].append(class_emo[0]['emotions']['disgust'])

Dict['fear'].append(class_emo[0]['emotions']['fear'])

Dict['happy'].append(class_emo[0]['emotions']['happy'])

Dict['sad'].append(class_emo[0]['emotions']['sad'])

Dict['surprise'].append(class_emo[0]['emotions']['surprise'])

Dict['neutral'].append(class_emo[0]['emotions']['neutral'])
            print(index_time_stamp)
```

```
            index_time_stamp += 1

    cv2.waitKey(0)

csv_path =
r'C:\Users\Mirus\PycharmProjects\Thesis\CSV_imagenes\S3_emoti
ons_score_time_stamp.csv'
(pd.DataFrame(Dict)).to_csv(csv_path, index=False,
header=True)
```

# 2. PROCESAMIENTO Y EXTRACCIÓN DE CARACTERÍSTICAS DE LA SEÑAL DE ACTIVIDAD ELECTRODÉRMICA

```
import preprocessing as preprocessing
import pyxdf
from sklearn import preprocessing
import matplotlib.pyplot as plt
import numpy as np
import scipy
from scipy import signal
import neurokit2 as nk
from scipy.signal import correlate
import pandas as pd

# dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0020\ses-
martes\eeg\sub-P0020_ses-martes_task-Default_run-
001_eeg.xdf")
# dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0023\ses-
martes\eeg\sub-P0023_ses-martes_task-Default_run-
001_eeg.xdf")
dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0026\ses-
miercoles\eeg\sub-P0026_ses-miercoles_task-Default_run-
001_eeg.xdf")

eda_series0 = []
gsr_series = []
cam_series = []

Dict_e4 = {'Signal_e4': [], 'Onsets_e4': [], 'Peaks_e4': [],
'Height_e4': [], 'Amplitude_e4': []}
Dict_mod = {'Signal_mod': [], 'Onsets_mod': [], 'Peaks_mod':
[], 'Height_mod': [], 'Amplitude_mod': []}

for stream in dataf:
    for i in range(len(dataf)):
        r = dataf[i]['info']['name']
        if r == ['eda']:
            eda_series0 = (dataf[i]['time_series'])
            eda_stamp = (dataf[i]['time_stamps'])
        else:
            pass
```

```
        if r == ['gsr']:
            gsr_series = (dataf[i]['time_series'])
            gsr_stamp = (dataf[i]['time_stamps'])
        else:
            pass
        if r == ['cam']:
            start_restriction = (dataf[i]['time_series'])[0]
        else:
            pass
eda_series = eda_series0[:,1:2]

def func_upsample_prefilter(filtered, series, stamp):
    (upsample_signal, upsample_t) =
scipy.signal.resample(filtered,
len(np.array(series.flat))*25, t=stamp, axis=0)
    fs = 100
    fc = 12.5
    w = fc * 2 / fs
    b, a = scipy.signal.butter(4, w, 'low')
    filtered_signal =
scipy.signal.filtfilt(b,a,np.array(upsample_signal).flat)
    return filtered_signal
filt_post_up_gsr =
func_upsample_prefilter(gsr_series,gsr_series,gsr_stamp)
filt_post_up_eda =
func_upsample_prefilter(eda_series,eda_series,eda_stamp)

def func_downsample(filtered, series):
    return scipy.signal.resample(filtered,
int(len(np.array(series.flat))*0.125), axis=0)
def func_post_filter(filtered,fc):
    fs = 12.5
    w = fc * 2 / fs
    b, a = scipy.signal.butter(5, w, 'low')
    filtered_signal =
scipy.signal.filtfilt(b,a,np.array(filtered).flat)
    return filtered_signal

filtered_eda =
func_post_filter(func_downsample(filt_post_up_eda,filt_post_u
p_eda),fc=0.75)
filtered_gsr =
func_post_filter(func_downsample(filt_post_up_gsr,filt_post_u
p_gsr),fc=1)
```

```
def func_normalization_n_window(signal0):
    signal1 = np.array(np.array(signal0).flat)
    normalized_signal =
preprocessing.normalize((signal1.reshape(-1,
1))[int(start_restriction):19389], axis=0)
    return normalized_signal
norm_gsr = func_normalization_n_window(filtered_gsr)
norm_eda =
np.fliplr(func_normalization_n_window(filtered_eda)*(-1))

def func_corr_function(x, y):
    N = max(len(x), len(y))
    n = min(len(x), len(y))
    if N == len(y):
        lags = np.arange(-N + 1, n)
    else:
        lags = np.arange(-n + 1, N)
    correl = correlate(x / np.std(x), y / np.std(y), 'full')
    return lags, correl, n

def func_alineacion(señal_a,señal_b):
    señalgsr = señal_a-señal_a[0]
    señaleda = señal_b-señal_b[0]
    return (señalgsr,señaleda)
(aligned_gsr,aligned_eda)= func_alineacion(norm_gsr,norm_eda)

lags, correl, n = func_corr_function(aligned_gsr,
aligned_eda)
numberofcorr = np.max(correl/n)
max_numer = [i for i,x in enumerate(correl/n) if x ==
numberofcorr]
position_sample = lags[:1]+max_numer

num = 12.5
tiempo_sec=3*60+30
largo = int(19389-start_restriction)
try_signal_e4 =
np.array(np.concatenate(([np.zeros(abs(int(position_sample)))
],[aligned_gsr]), axis=None).flat)
try_signal_mod = np.array(aligned_eda.flat)

def func_window(signal,largo,position_sample):
    return signal[0 +
abs(int(position_sample)):abs(int(largo)) +
abs(int(position_sample))]
```

```
e4_signal =
np.array(func_window(try_signal_e4,largo,position_sample).fla
t)
mod_signal =
np.array(func_window(try_signal_mod,largo,0).flat)

for arr in e4_signal.reshape(-1, 1):
    i = arr[0]
    Dict_e4['Signal_e4'].append([i])

for arr in mod_signal.reshape(-1, 1):
    i = arr[0]
    Dict_mod['Signal_mod'].append([i])

def func_extract_phasic(signal,num):
    return
nk.eda_phasic(nk.standardize(signal),sampling_rate=num)
e4_data = func_extract_phasic(e4_signal,num)
mod_data = func_extract_phasic(mod_signal,num)

def func_valores_tonic_phasic(signal):
    t,p = signal["EDA_Tonic"].values,
signal["EDA_Phasic"].values
    return (t,p)
(t0, p0) = func_valores_tonic_phasic(e4_data)
(t1, p1) = func_valores_tonic_phasic(mod_data)

e4_data['EDA_Raw'] = e4_signal
mod_data['EDA_Raw'] = mod_signal

def func_peaks_onset_AmHt(p,num):
    signal_findpeak = nk.eda_findpeaks(p, sampling_rate=num)
    calc_threshold = ((signal_findpeak['SCR_Height'].max()-
signal_findpeak['SCR_Height'].min())/2)*0.01
    peaks_array = nk.eda_peaks(p, sampling_rate=num,
method='kim2004', amplitude_min=calc_threshold) #neurokit,
gamboa2008, kim2004, vanhalem2020, nabian2018
    final_peaks =
nk.eda_fixpeaks(peaks_array[0]["SCR_Peaks"],
onsets=peaks_array[0]["SCR_Onsets"])
    return peaks_array, final_peaks,
peaks_array[0]["SCR_Onsets"]
fp_e4_for_ampl,SCR_Peaks_e4, SCR_Onsets_e4  =
func_peaks_onset_AmHt(p0, num)
fp_mod_for_ampl,SCR_Peaks_mod, SCR_Onsets_mod =
```

```
func_peaks_onset_AmHt(p1, num)

for arr in np.array(SCR_Onsets_e4).reshape(-1, 1):
    i = arr[0]
    Dict_e4['Onsets_e4'].append([i])
for arr in np.array(SCR_Peaks_e4['SCR_Peaks']).reshape(-1,
1):
    i = arr[0]
    Dict_e4['Peaks_e4'].append([i])

for arr in np.array(SCR_Onsets_mod).reshape(-1, 1):
    i = arr[0]
    Dict_mod['Onsets_mod'].append([i])
for arr in np.array(SCR_Peaks_mod['SCR_Peaks']).reshape(-1,
1):
    i = arr[0]
    Dict_mod['Peaks_mod'].append([i])

def func_height_ampl(signal):
    height = signal[0]["SCR_Height"]
    ampl = signal[0]["SCR_Amplitude"]
    return height, ampl
SCR_Height_e4, SCR_Ampl_e4 = func_height_ampl(fp_e4_for_ampl)
SCR_Height_mod, SCR_Ampl_mod =
func_height_ampl(fp_mod_for_ampl)

for arr in np.array(SCR_Height_e4).reshape(-1, 1):
    i = arr[0]
    Dict_e4['Height_e4'].append([i])
for arr in np.array(SCR_Ampl_e4).reshape(-1, 1):
    i = arr[0]
    Dict_e4['Amplitude_e4'].append([i])

for arr in np.array(SCR_Height_mod).reshape(-1, 1):
    i = arr[0]
    Dict_mod['Height_mod'].append([i])
for arr in np.array(SCR_Ampl_mod).reshape(-1, 1):
    i = arr[0]
    Dict_mod['Amplitude_mod'].append([i])

plt.plot(e4_signal, label='E4 señal', color="r")
plt.plot(mod_signal, label='Mod señal', color="b")
plt.plot(np.array(SCR_Peaks_e4['SCR_Peaks'])*e4_signal,
label='Picos de SCR E4', color="r")
plt.plot(np.array(SCR_Peaks_mod['SCR_Peaks'])*mod_signal,
```

```
label='Picos de SCR Mod', color="b")
# plt.plot(np.array(SCR_Onsets_e4)*e4_signal, label='Onsets
E4',color="orange")
# plt.plot(np.array(SCR_Onsets_mod)*mod_signal, label='Onsets
Mod', color="c")
plt.legend()
#plt.show()

csv_pathE4 =
r'C:\Users\Mirus\PycharmProjects\Thesis\CSV_EDA_MOD\S4_E4.csv
'
csv_pathMSWH =
r'C:\Users\Mirus\PycharmProjects\Thesis\CSV_EDA_MOD\S4_MSWH.c
sv'

(pd.DataFrame(Dict_e4)).to_csv(csv_pathE4, index=False,
header=True)
(pd.DataFrame(Dict_mod)).to_csv(csv_pathMSWH, index=False,
header=True)
```

# 3. PROCESAMIENTO Y EXTRACCIÓN DE CARACTERÍSTICAS DE LA FRECUENCIA CARDÍACA

```
import math
import pyxdf
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy
from scipy import signal

# dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0020\ses-
martes\eeg\sub-P0020_ses-martes_task-Default_run-
001_eeg.xdf")
# dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0023\ses-
martes\eeg\sub-P0023_ses-martes_task-Default_run-
001_eeg.xdf")
dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0026\ses-
miercoles\eeg\sub-P0026_ses-miercoles_task-Default_run-
001_eeg.xdf")

sampling_rate = 12.5

for stream in dataf:
    for i in range(len(dataf)):
        r = dataf[i]['info']['name']
        if r == ['ibi']:
            ibi_series0 = (dataf[i]['time_series'])
            ibi_stamp0 = (dataf[i]['time_stamps'])
        else:
            pass
        if r == ['cam']:
            start_restriction = (dataf[i]['time_series'])[0]
        else:
            pass

Dict_HR = {'Signal_IBI': [],
           'Signal_HR': [],
           'Peaks': [],
           'Standard_Dev': []}

ibi_series = ibi_series0
ibi_stamp = ibi_stamp0
```

```
SD_2_array = []

def func_resample(series, stamp, orden, fs, fc, val_float):
    signal_hr = []
    (resample_signal, resample_t) =
scipy.signal.resample(np.array(series).flat,int(len(np.array(
(np.array(series)).flat)) * val_float), t=stamp, axis=0)
    w = fc * 2 / fs
    b, a = scipy.signal.butter(orden, w, 'low')
    signal_ibi=scipy.signal.filtfilt(b, a,
np.array(resample_signal).flat)
    for arri in (signal_ibi.reshape(-1, 1)):
        i = arri[0]
        signal_hr.append((60 / i))
    return signal_ibi, signal_hr
filtered_ibi, hr_series = func_resample(series=ibi_series,
stamp=ibi_stamp, orden=5, fs=sampling_rate, fc=0.35,
val_float=8.7175 )

def func_windows(signal):
    return signal[int(start_restriction):19389]

ventana_ibi = func_windows(signal=filtered_ibi)
ventana_hr = func_windows(signal=hr_series)

for arr in np.array(ventana_ibi).reshape(-1, 1):
    i = arr[0]
    Dict_HR['Signal_IBI'].append([i])

for arr in np.array(ventana_hr).reshape(-1, 1):
    i = arr[0]
    Dict_HR['Signal_HR'].append([i])


def func_Standard_Deviation(signal0):
    signal = np.array(np.array(signal0).flat)
    for i in range(len(signal) - 1):
        values = np.std([signal[i], signal[i+1]])
        SD_2_array.append(values)
    return SD_2_array
SD_HR =
np.array(np.concatenate(([np.zeros(abs(int(1)))],[func_Standa
rd_Deviation(signal0=ventana_hr)]), axis=None).flat)

for arr in np.array(SD_HR).reshape(-1, 1):
```

```
    i = arr[0]
    Dict_HR['Standard_Dev'].append([i])

def func_mean_HR(signal, num_data_points):
    return
np.sum(np.array(np.array(signal).flat))/num_data_points
mean_HR = func_mean_HR(signal=ventana_hr,
num_data_points=len(ventana_hr))

def calc_umbral(mean_HR):
    porcentaje = 0.8
    umbral_per=int(mean_HR*porcentaje)
    return umbral_per
umbral_per=calc_umbral(mean_HR=mean_HR)

def func_peaks(signal, umbral):
    peaks = []
    for i in range(1, len(signal) - 1):
        if signal[i] > signal[i-1] and signal[i] >
signal[i+1] and signal[i] > umbral:
            peaks.append(1)
        else:
            peaks.append(0)
    return peaks
peaks_per =
np.concatenate([np.array(func_peaks(signal=ventana_hr,umbral=
umbral_per)), np.zeros(2)])

for arr in peaks_per.reshape(-1, 1):
    i = arr[0]
    Dict_HR['Peaks'].append([i])

plt.plot(peaks_per[:2000]*105)
plt.plot(ventana_hr[:2000])
plt.show()

csv_pathHR =
r'C:\Users\Mirus\PycharmProjects\Thesis\CSV_HeartRate\S4_HRV.
csv'

(pd.DataFrame(Dict_HR)).to_csv(csv_pathHR, index=False,
header=True)
```

# 4. READAPTACIÓN DE SEÑALES

```
import pyxdf
import matplotlib.pyplot as plt
import numpy as np
import scipy
from scipy import signal
import pandas as pd

dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0020\ses-
martes\eeg\sub-P0020_ses-martes_task-Default_run-
001_eeg.xdf")
# dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0021\ses-
martes\eeg\sub-P0021_ses-martes_task-Default_run-
001_eeg.xdf")
# dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0023\ses-
martes\eeg\sub-P0023_ses-martes_task-Default_run-
001_eeg.xdf")
# dataf, header = pyxdf.load_xdf(r"D:\Datos\sub-P0026\ses-
miercoles\eeg\sub-P0026_ses-miercoles_task-Default_run-
001_eeg.xdf")

new_arr = []
Dict_answers = {'Answer_people': []}
Dict_videoID = {'Sequence_videos': []}

for stream in dataf:
    for i in range(len(dataf)):
        r = dataf[i]['info']['name']
        if r == ['emo6BDATA']:
            videoAns_series =
np.array(np.array((dataf[i]['time_series'])[:,1]).flat)
            videoAns_stamp = (dataf[i]['time_stamps'])
        else:
            pass
        if r == ['emo6BIDs']:
            Dsy_series0 = (dataf[i]['time_series'])
            Dsy_stamp = (dataf[i]['time_stamps'])
        else:
            pass
        if r == ['cam']:
            start_restriction = (dataf[i]['time_series'])[0]
        else:
            pass
```

```
for i in (np.array(Dsy_series0)[:, 1]).reshape(-1, 1):
    idx = i[0]
    new_arr.append([int(idx)])
Dsy_series = np.array(np.array(new_arr).flat)
bmark_ppl_answer = []
bmark_sequence_answer = []

for arr in np.array(videoAns_series).reshape(-1, 1):
    i = [arr[0]]
    if arr[0] == 99.0:
        bmark_ppl_answer.append(6)
    if arr[0] == 11.0:
        bmark_ppl_answer.append(1)
    if arr[0] == 21.0:
        bmark_ppl_answer.append(3)
    if arr[0] == 31.0:
        bmark_ppl_answer.append(0)
    if arr[0] == 41.0:
        bmark_ppl_answer.append(2)
    if arr[0] == 51.0:
        bmark_ppl_answer.append(5)
    if arr[0] == 61.0:
        bmark_ppl_answer.append(4)
    if arr[0] == 32.0:
        bmark_ppl_answer.append(3)
    if arr[0] == 22.0:
        bmark_ppl_answer.append(3)
    if arr[0] == 52.0:
        bmark_ppl_answer.append(1)
    if arr[0] == 12.0:
        bmark_ppl_answer.append(1)
    if arr[0] == 62.0:
        bmark_ppl_answer.append(4)
    if arr[0] == 42.0:
        bmark_ppl_answer.append(5)
    if arr[0] == 53.0:
        bmark_ppl_answer.append(3)
    if arr[0] == 43.0:
        bmark_ppl_answer.append(2)
    if arr[0] == 33.0:
        bmark_ppl_answer.append(3)
    if arr[0] == 23.0:
        bmark_ppl_answer.append(3)
    if arr[0] == 13.0:
        bmark_ppl_answer.append(1)
    else:
        pass
```

```
for arr in np.array(Dsy_series).reshape(-1, 1):
    i = [arr[0]]
    if arr[0] == 0.0:
        bmark_sequence_answer.append(6)
    if arr[0] == 11.0:
        bmark_sequence_answer.append(1)
    if arr[0] == 21.0:
        bmark_sequence_answer.append(3)
    if arr[0] == 31.0:
        bmark_sequence_answer.append(0)
    if arr[0] == 41.0:
        bmark_sequence_answer.append(2)
    if arr[0] == 51.0:
        bmark_sequence_answer.append(5)
    if arr[0] == 61.0:
        bmark_sequence_answer.append(4)
    if arr[0] == 32.0:
        bmark_sequence_answer.append(0)
    if arr[0] == 22.0:
        bmark_sequence_answer.append(3)
    if arr[0] == 52.0:
        bmark_sequence_answer.append(5)
    if arr[0] == 12.0:
        bmark_sequence_answer.append(1)
    if arr[0] == 62.0:
        bmark_sequence_answer.append(4)
    if arr[0] == 42.0:
        bmark_sequence_answer.append(2)
    if arr[0] == 53.0:
        bmark_sequence_answer.append(5)
    if arr[0] == 43.0:
        bmark_sequence_answer.append(2)
    if arr[0] == 33.0:
        bmark_sequence_answer.append(0)
    if arr[0] == 23.0:
        bmark_sequence_answer.append(3)
    if arr[0] == 13.0:
        bmark_sequence_answer.append(1)
    else:
        pass

def func_resample(signal0,stamp):
    signal = np.array(np.array(signal0).flat)
    resample_signal, _ = scipy.signal.resample(signal,
int(len(np.array(signal.flat))), t=stamp, axis=0)
    new_indices = np.linspace(0, len(resample_signal) - 1,
```

```
int(len(resample_signal)*7.63804803))
    interpolated_array = np.interp(new_indices,
np.arange(len(resample_signal)), resample_signal)
    return np.round(interpolated_array).astype(np.int32)
bmark_ANSWER = func_resample(bmark_ppl_answer,
videoAns_stamp)
bmark_ID = func_resample(bmark_sequence_answer, Dsy_stamp)

def func_window(signal, init_val):
    return np.array(np.array(signal).flat)[int(init_val):]

benchmark_people =
func_window(bmark_ANSWER,start_restriction)
benchmark_videos = func_window(bmark_ID, start_restriction)
plt.plot(bmark_sequence_answer[:250], label='sequence')
plt.plot(bmark_ppl_answer[:250], label='answers')
plt.legend()
plt.show()

for arr in benchmark_videos.reshape(-1, 1):
    i = arr[0]
    Dict_videoID['Sequence_videos'].append([i])

for arr in benchmark_people.reshape(-1, 1):
    i = arr[0]
    Dict_answers['Answer_people'].append([i])
csv_path =
r'C:\Users\Mirus\PycharmProjects\Thesis\CSV_Answers_&_ID\S1_A
nswers.csv'
csv_pathID =
r'C:\Users\Mirus\PycharmProjects\Thesis\CSV_Answers_&_ID\S1_V
ideoID.csv'

(pd.DataFrame(Dict_answers)).to_csv(csv_path, index=False,
header=True)
(pd.DataFrame(Dict_videoID)).to_csv(csv_pathID, index=False,
header=True)
```

# 5. RED NEURONAL BIDIRECCIONAL DE MEMORIA A LARGO Y CORTO PLAZO (MANUAL)

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split,
GridSearchCV
from tensorflow.keras.layers import Input, Conv1D,
MaxPooling1D, Flatten, Dense, Concatenate
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Bidirectional,
Dense
from tensorflow.keras.wrappers.scikit_learn import
KerasClassifier
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import accuracy_score

labels_ =
pd.read_csv(r'C:\Users\Mirus\PycharmProjects\Thesis\CSV_LABEL
S.csv')['Video']
#labels_ = labels_0.drop('Video', axis=1)
features_ =
pd.read_csv(r'C:\Users\Mirus\PycharmProjects\Thesis\CSV_FEATU
RES.csv')

X_train = features_[:27286]
X_test = features_[27286:]

y_train = to_categorical(labels_[:27286]) #labels_[:27286]
y_test = to_categorical(labels_[27286:]) #labels_[27286:]

input_shape = (features_.shape[1], 1)
num_classes = len(labels_.unique()) #labels_.shape[1]

model = Sequential()
model.add(Bidirectional(LSTM(units=64,
return_sequences=True), input_shape=input_shape))
model.add(Bidirectional(LSTM(units=64)))
model.add(Dense(units=num_classes, activation='softmax'))

model.compile(optimizer='adamax',
loss='categorical_crossentropy', metrics=['accuracy'])
```

```
model.fit(X_train, y_train, batch_size=32, epochs=30,
validation_split=0.2)
loss, accuracy = model.evaluate(X_test, y_test)

predictions = model.predict(X_test)

csv_pathHR =
r'C:\Users\Mirus\PycharmProjects\Thesis\CSV_HeartRate\predict
ions_videoid_me.csv'

(pd.DataFrame(predictions)).to_csv(csv_pathHR, index=False,
header=True)
```

## 5.1. OPTIMIZACIÓN AUTOMATIZADA

```
def create_model(units, activation, optimizer, loss):
    model = Sequential()
    model.add(Bidirectional(LSTM(units=units,
return_sequences=True), input_shape=input_shape))
    model.add(Bidirectional(LSTM(units=units)))
    model.add(Dense(units=num_classes,
activation=activation))

    model.compile(optimizer=optimizer, loss=loss,
metrics=['accuracy'])
    return model

model = KerasClassifier(build_fn=create_model, verbose=0)

param_grid = {
    'units': [16, 32, 64],
    'activation': ['softmax', 'relu', 'sigmoid'],
    'optimizer': ['adam', 'RMSprop'],
    'loss': ['categorical_crossentropy',
'kullback_leibler_divergence']
}
grid = GridSearchCV(estimator=model, param_grid=param_grid,
cv=3)
grid_result = grid.fit(X_train, y_train)
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
best_model = grid_result.best_estimator_
y_prediction = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_prediction)
```

(_) Memoria 109 páginas

(X) Anexos 24 páginas

La Almunia, a 05 de 09 de 2023

Firmado por IEMBERGENOVA ILMIRA - ****0426* el día 02/09/2023 con un certificado emitido por AC FNMT Usuarios

Firmado: Ilmira Iembergenova