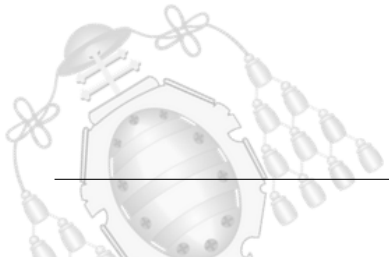
B. Código fuente

B.1.- Programa de entrenamiento modelo VGG

```
# LIBRERÍAS
import os
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true' #Evita complicaciones con CUDA (Si
    no lo utilizas, coméntala)
#os.environ['CUDA_VISIBLE_DEVICES'] = '-1' Descomenta esta línea para forzar el uso
    de CPU. Si no usas CUDA, siempre se ejecutará por CPU por lo que no es necesaria
import sys
import subprocess
from subprocess import call
import numpy as np
import pandas as pd
import tensorflow as tf
import keyboard
import traceback
import datetime
from os import remove
import tensorflow.compat.v1.keras.utils as np_utils
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.python.keras.applications.vgg19 import VGG19
from tensorflow.keras.layers import Dense, Dropout, Input, Flatten, Conv2D,
    MaxPooling2D, Activation, BatchNormalization
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adamax
from tensorflow.keras import utils
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
from sklearn import metrics, decomposition
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
```



```
# PARÁMETROS
PRE_TRAINED_MODEL = "VGG16"
IMG_SIZE = 48
EPOCH = 10000
NUM_TRAIN = 28708
NUM_VALIDATION = 7178
NUM_CLASSES = 7
BATCH = 64
LEARNING_RATE = 0.0005
DECAY = 1e-5
PATIENCE = 50

# DIRECCTORIOS
CSV_FILE= 'dataset.csv'
TIME = datetime.datetime.now().strftime("%Y%m%d-%H_%M_%S")
SESSION_PATH= "run_" + TIME +"\\"
LOG_CSV = SESSION_PATH + "log_csv" + ".log"
LOGS = "logs\\"
FIT = "logs\\"+ "fit\\"
LOG_DIR = FIT+ TIME
CKPT_DIR = SESSION_PATH+ "ckpt\\"

if not os.path.exists(SESSION_PATH):
    os.mkdir(SESSION_PATH)

if not os.path.exists(CKPT_DIR):
    os.mkdir(CKPT_DIR)

if not os.path.exists(LOGS):
    os.mkdir(LOGS)

if not os.path.exists(FIT):
    os.mkdir(FIT)

if not os.path.exists(LOG_DIR):
    os.mkdir(LOG_DIR)
```

```
#VERSIONES
print("TensorFlow version: ", tf.__version__)
print("Versión de Python: ",sys.version)

# CLASE
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(epoch % 5 == 0) :
            files_to_delete = os.listdir(SESSION_PATH + "ckpt\\")
            # Ordenamos la lista de ficheros alfabeticamente
            files_to_delete.sort(reverse=True)
            # Solo puede haber dos ficheros, csv.log y el mejor checkpoint
            while(len(files_to_delete)>1):
                # Se borra el archivo
                try:
                    remove(SESSION_PATH + "ckpt\\" + files_to_delete[1])
                    del(files_to_delete[1])
                except Exception:
                    print("Un archivo aún no ha podido ser eliminado, se eliminará más
                        tarde")

        if keyboard.is_pressed(''):
            print('Se aborto manualmente el entrenamiento')
            self.model.stop_training = True



#FUNCIONES
def load_data(csv_dir):
    ### CARGAR DATOS DEL CSV
    # Todo los datos
    print('Comienza la importación de los datos')
    # Análisis de los datos
    data_csv = pd.read_csv(csv_dir, header=0)
    print(data_csv.Usage.value_counts())
```

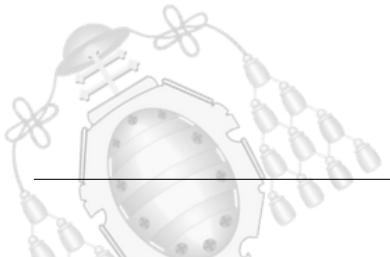
```
    print('La base de datos contiene {} caras'.format(data_csv.shape[0]))
    emotion_list = list(data_csv.emotion)
    pixels_list = list(data_csv.pixels)

    # Se transforman las emociones a un array categorical
    emotion_array_np = np_utils.to_categorical(np.asarray(emotion_list),
        num_classes=NUM_CLASSES, dtype='float32')
    images_array_np = create_images_array(pixels_list)
    return emotion_array_np, images_array_np


def create_images_array(pixels_list):
        # Construimos un vector de matrices, donde cada matriz será una foto
        images_array = []
        for image in pixels_list:

                image_matrix = np.zeros((IMG_SIZE, IMG_SIZE), dtype=np.uint8)
                pixels_array = image.split()

                for row_num in range(IMG_SIZE):
                        start_row = row_num * IMG_SIZE
                        finish_row = start_row + IMG_SIZE
                        image_matrix[row_num] = pixels_array[ start_row : finish_row]

                images_array.append(np.array(image_matrix))

        images_array_np = (np.array(images_array)).astype('float32') / 255.0
        return images_array_np


def apply_PreTrained_model(model,data):
    #Se crea el modelo pre entrenado selecionado
    #include_top=False no incluye las ultimas capas softmax
    if model== "VGG16":
        CNN_model = VGG16(include_top=False, input_shape=(48, 48, 3), pooling='avg',
            weights='imagenet')
```

```
    if model== "VGG19":
        CNN_model = VGG19(include_top=False, input_shape=(48, 48, 3), pooling='avg',
            weights='imagenet')

    result = CNN_model.predict(data)
    #Se aplica a los datos y se retorna el modelo para contruir el modelo final
        completo
    return result, CNN_model


def main():
    # Obtenemos dos numpy array con las emociones y las imagenes en orden
    emotion_array_np, images_array_np = load_data(CSV_FILE)

    # Triplicamos las imágenes a 3 capas, para tenerlas en formato RGB,
        shape(NUM,48,48) --> shape(NUM,48,48,3)
    images_array_np =
        np.broadcast_to(images_array_np[...,None],images_array_np.shape+(3,))
    images_original_np = images_array_np

    images_train_original = images_original_np[0:NUM_TRAIN]
    images_test_original = images_original_np[NUM_TRAIN+1:]

    # Guardamos el formato de entrada al modelo
    initial_inputs = Input(np.shape(images_array_np[0]))
    images_array_np, PreTrained_Model =
        apply_PreTrained_model(PRE_TRAINED_MODEL,images_array_np)

    # FORMATO ENTRADA shape(NUM,48,48,3)
    # SE DIVIDE EN:
    # TRAIN
    emotion_train = emotion_array_np[0:NUM_TRAIN]
    images_train = images_array_np[0:NUM_TRAIN]

    # TEST
    emotion_test = emotion_array_np[NUM_TRAIN+1:]
    images_test = images_array_np[NUM_TRAIN+1:]
```

```
# Eliminamos el csv inicial
del (images_array_np, emotion_array_np)


#Construimos la red neuronal
neuronal_network = Sequential()

#Completamos el modelo con nuestras capas
# Capa input
neuronal_network.add(Dense(256, input_shape= np.shape(images_train[0]) ,
    activation='relu'))
neuronal_network.add(Dropout(0.25))
# Capa 1
neuronal_network.add(Dense(128, input_shape= (256,) , activation='relu'))
neuronal_network.add(Dropout(0.25))
# Capa 2
neuronal_network.add(Dense(64, input_shape=(128,)))
neuronal_network.add(Dropout(0.25))
# Capa Out
neuronal_network.add(Dense(NUM_CLASSES, activation='softmax'))

neuronal_network.compile(loss='categorical_crossentropy',
               optimizer=Adamax(lr= LEARNING_RATE, decay =
                   DECAY),metrics=['accuracy'])


neuronal_network.compile(loss='categorical_crossentropy', optimizer=Adamax(lr=
    LEARNING_RATE, decay= DECAY),metrics=['accuracy'])

# Gráficos a timepo real
# Se define tensorboard para seguir el entrenamiento a tiempo real
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=LOG_DIR)

try:
    # Lanzamos los tensorboard
```

```
    process = subprocess.Popen('tensorboard --logdir=./logs',
        stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    # Abrimos el navegador
    os.system('start chrome http://localhost:6006/')
except Exception:
    print ("No pudo lanzarse tensorboard. Excepción: \n", traceback.print_exc())


# CALLBACKS
checkpoint_name = 'val_acc_{val_accuracy:.4f}-{epoch:02d}.hdf5'
checkpoint_filepath = CKPT_DIR + checkpoint_name
csv_logger = tf.keras.callbacks.CSVLogger(LOG_CSV, append=False)
# early_stop = tf.keras.callbacks.EarlyStopping('val_loss', patience=PATIENCE)
model_checkpoint = tf.keras.callbacks.ModelCheckpoint(checkpoint_filepath,
    monitor='val_accuracy', verbose=1, mode='max', save_best_only=True)
callbacks = [myCallback(), tensorboard, model_checkpoint, csv_logger]

# ENTRENAMIENTO
model_info = neuronal_network.fit(
        images_train,emotion_train,
        batch_size= BATCH,
        epochs=EPOCH,
        validation_data= (images_test,emotion_test),
        verbose=1,
        callbacks= callbacks
        )

#CONSTRUCCIÓN DEL MODELO COMPLETO
neuronal_input = PreTrained_Model(initial_inputs)
model_output = neuronal_network(neuronal_input)

#Pre trained model + neuronal_network
convolutional_model = Model( initial_inputs, model_output)
convolutional_model.compile(loss='categorical_crossentropy', optimizer=Adamax(),
    metrics=['accuracy'])

# GUARDADO DE LA ESTRUCTURA Y CONFIGURACIÓN EN UN .txt
```

```
    file = open(SESSION_PATH + "info_model.txt", "w")
    convolutional_model.summary(print_fn=lambda x: file.write(x + '\n'))
    file.write("\n\n" + "Learning rate: " + str(LEARNING_RATE) + "\nDecay: " +
        str(DECAY))
    file.close()

    #Evaluamos el modelo final
    results = neuronal_network.evaluate(images_test, emotion_test, batch_size=128)

    # GUARDADO DEL MODELO
    accuracy = model_info.history['accuracy']
    epochs = str(len(accuracy))
    a = round(results[1], 3)
    accuracy_value = str(a)

    model_name = "model_e_"+ epochs + "-acc_" + accuracy_value +".hdf5"
    try:
        convolutional_model.save(SESSION_PATH + model_name )
    except Exception:
        print("Error al guardar el modelo \n", traceback.print_exc())


if __name__ == "__main__":
    main()
```



B.2.- Programa de entrenamiento modelo CNN

```
# LIBRERÍAS
import os
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true' #Evita complicaciones con CUDA (Si
    no lo utilizas, coméntala)
#os.environ['CUDA_VISIBLE_DEVICES'] = '-1' Descomenta esta línea para forzar el uso
    de CPU. Si no usas CUDA, siempre se ejecutará por CPU por lo que no es necesaria
import sys
import subprocess
from subprocess import call
import numpy as np
import pandas as pd
import tensorflow as tf
import keyboard
import traceback
import datetime
from os import remove
import tensorflow.compat.v1.keras.utils as np_utils
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.python.keras.applications.vgg19 import VGG19
from tensorflow.keras.layers import Dense, Dropout, Input, Flatten, Conv2D,
    MaxPooling2D, Activation, BatchNormalization
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adamax
from tensorflow.keras import utils
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
from sklearn import metrics, decomposition
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix



# PARÁMETROS
PRE_TRAINED_MODEL = "VGG16"
IMG_SIZE = 48
EPOCH = 10000
NUM_TRAIN = 28708
```

```
NUM_VALIDATION = 7178
NUM_CLASSES = 7
BATCH = 64
LEARNING_RATE = 0.0005
DECAY = 1e-5
PATIENCE = 50

# PARÁMETROS CONVOLUCIÓN
NUM_CONV1 = 32
NUM_CONV2= 64
NUM_CONV3= 128
NUM_CONV4 = 128
TAM_FILTER = (3,3)


# DIRECCTORIOS
CSV_FILE= 'dataset.csv'
TIME = datetime.datetime.now().strftime("%Y%m%d-%H_%M_%S")
SESSION_PATH= "run_" + TIME +"\\"
LOG_CSV = SESSION_PATH + "log_csv" + ".log"
LOGS = "logs\\"
FIT = "logs\\"+ "fit\\"
LOG_DIR = FIT+ TIME
CKPT_DIR = SESSION_PATH+ "ckpt\\"

if not os.path.exists(SESSION_PATH):
    os.mkdir(SESSION_PATH)

if not os.path.exists(CKPT_DIR):
    os.mkdir(CKPT_DIR)

if not os.path.exists(LOGS):
    os.mkdir(LOGS)

if not os.path.exists(FIT):
    os.mkdir(FIT)
```

```
if not os.path.exists(LOG_DIR):
    os.mkdir(LOG_DIR)



#VERSIONES
print("TensorFlow version: ", tf.__version__)
print("Versión de Python: ",sys.version)

# CLASE
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(epoch % 5 == 0) :
            files_to_delete = os.listdir(SESSION_PATH + "ckpt\\")
            # Ordenamos la lista de ficheros alfabeticamente
            files_to_delete.sort(reverse=True)
            # Solo puede haber dos ficheros, csv.log y el mejor checkpoint
            while(len(files_to_delete)>1):
                # Se borra el archivo
                try:
                    remove(SESSION_PATH + "ckpt\\" + files_to_delete[1])
                    del(files_to_delete[1])
                except Exception:
                    print("Un archivo aún no ha podido ser eliminado, se eliminará más
                        tarde")

        if keyboard.is_pressed(''):
            print('Se aborto manualmente el entrenamiento')
            self.model.stop_training = True



# FUNCIONES
def load_data(csv_dir):
    # CARGAR DATOS DEL CSV
    # Todos los datos
    print('Comienza la importación de los datos')
    # Análisis de los datos
```

```
    data_csv = pd.read_csv(csv_dir, header=0)
    print(data_csv.Usage.value_counts())
    print('La base de datos contiene {} caras'.format(data_csv.shape[0]))
    emotion_list = list(data_csv.emotion)
    pixels_list = list(data_csv.pixels)

    # Se transforman las emociones a un array categorical
    emotion_array_np = np_utils.to_categorical(np.asarray(emotion_list),
        num_classes=NUM_CLASSES, dtype='float32')
    images_array_np = create_images_array(pixels_list)
    return emotion_array_np, images_array_np


def create_images_array(pixels_list):
        # Construimos un vector de matrices, donde cada matriz será una foto
        images_array = []
        for image in pixels_list:

                image_matrix = np.zeros((IMG_SIZE, IMG_SIZE), dtype=np.uint8)
                pixels_array = image.split()

                for row_num in range(IMG_SIZE):
                        start_row = row_num * IMG_SIZE
                        finish_row = start_row + IMG_SIZE
                        image_matrix[row_num] = pixels_array[ start_row : finish_row]

                images_array.append(np.array(image_matrix))

        images_array_np = (np.array(images_array)).astype('float32') / 255.0
        return images_array_np


def apply_PreTrained_model(model,data):
    # Se crea el modelo pre entrenado seleccionado
    # include_top=False no incluye las ultimas capas Softmax
    if model== "VGG16":
```

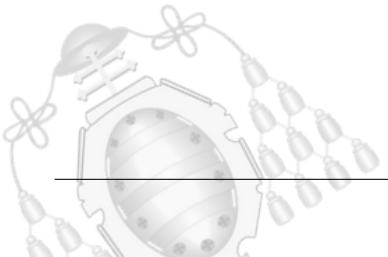
```
        CNN_model = VGG16(include_top=False, input_shape=(48, 48, 3), pooling='avg',
            weights='imagenet')

    if model== "VGG19":
        CNN_model = VGG19(include_top=False, input_shape=(48, 48, 3), pooling='avg',
            weights='imagenet')

    result = CNN_model.predict(data)
    # Se aplica a los datos y se retorna el modelo para contruir el modelo final
        completo
    return result, CNN_model


def main():
    # Obtenemos dos numpy array con las emociones y las imagenes en orden
    emotion_array_np, images_array_np = load_data(CSV_FILE)

    # Triplicamos las imagenes a 3 capas, para tenerlas en formato RGB,
        shape(NUM,48,48) --> shape(NUM,48,48,1)
    images_array_np =
        np.broadcast_to(images_array_np[...,None],images_array_np.shape+(1,))
    # Guardamos el formato de entrada al modelo
    initial_inputs = Input(np.shape(images_array_np[0]))

    # FORMATO ENTRADA shape(NUM,48,48,1)
    # SE DIVIDE EN:
    # TRAIN
    emotion_train = emotion_array_np[0:NUM_TRAIN]
    images_train = images_array_np[0:NUM_TRAIN]

    # TEST
    emotion_test = emotion_array_np[NUM_TRAIN+1:]
    images_test = images_array_np[NUM_TRAIN+1:]

    # Eliminamos el csv inicial
    del (images_array_np, emotion_array_np)
```

```
# Preprocesado de imágenes para crear nuevas imágenes
train_datagen = ImageDataGenerator(
    # Inclina y estira la imagen
    shear_range = 0.2,
    # Rotaciones aleatorias de 10 grados
    rotation_range=10,
    # Se aplica de forma aleatoria zoom en las imágenes
    zoom_range=0.2,
    # Algunas imágenes se girarán horizontalmente
    horizontal_flip=True,
    # Desplazamientos
    # Horizontal
    width_shift_range=0.1,
    # Vertical
    height_shift_range=0.1
)


# No hace ningún cambio, solo aplicará el nuevo formato con BATCH
test_datagen = ImageDataGenerator(
    horizontal_flip=False,
    zoom_range=0
)

data_train = train_datagen.flow(images_train, emotion_train, BATCH)
data_test = test_datagen.flow(images_test, emotion_test, BATCH)

# Generamos estructura secuencial, (capas apiladas)
CNN_model = Sequential()
# Capa convolucional 1
CNN_model.add(Conv2D(NUM_CONV1, kernel_size= TAM_FILTER, activation='relu',
    input_shape=np.shape(images_train[0])))

# Capa convolucional 2
CNN_model.add(Conv2D(NUM_CONV2, kernel_size= TAM_FILTER, activation='relu'))
CNN_model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
# Capa convolucional 3
CNN_model.add(Conv2D(NUM_CONV3, kernel_size= TAM_FILTER, activation='relu'))
CNN_model.add(MaxPooling2D(pool_size=(2, 2)))

# Capa convolucional 4
CNN_model.add(Conv2D(NUM_CONV4, kernel_size= TAM_FILTER, activation='relu'))
CNN_model.add(MaxPooling2D(pool_size=(2, 2)))

# Aplanamiento de los datos
CNN_model.add(Flatten())
# RED NEURONAL MULTICAPA
# Capa 1
CNN_model.add(Dense(2048, activation='relu'))
CNN_model.add(Dropout(0.5))
# Capa 2
CNN_model.add(Dense(1024, activation='relu'))
CNN_model.add(Dropout(0.5))
# Capa output
CNN_model.add(Dense(NUM_CLASSES, activation='softmax'))



CNN_model.compile(loss='categorical_crossentropy', optimizer=Adamax(lr=
    LEARNING_RATE, decay= DECAY),metrics=['accuracy'])

# Gráficos a tiempo real
# Se define tensorboard para seguir el entrenamiento e tiempo real
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=LOG_DIR)

try:
    # Lanzamos los tensorboard
    process = subprocess.Popen('tensorboard --logdir=./logs',
        stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    # Abrimos el navegador
    os.system('start chrome http://localhost:6006/')
except Exception:
    print ("No pudo lanzarse tensorboard. Excepción: \n", traceback.print_exc())
```

```
# CALLBACKS
checkpoint_name = 'val_acc_{val_accuracy:.4f}-{epoch:02d}.hdf5'
checkpoint_filepath = CKPT_DIR + checkpoint_name
csv_logger = tf.keras.callbacks.CSVLogger(LOG_CSV, append=False)
# early_stop = tf.keras.callbacks.EarlyStopping('val_loss', patience=PATIENCE)
model_checkpoint = tf.keras.callbacks.ModelCheckpoint(checkpoint_filepath,
    monitor='val_accuracy', verbose=1, mode='max', save_best_only=True)
callbacks = [myCallback(), tensorboard, model_checkpoint, csv_logger]

# ENTRENAMIENTO
model_info = CNN_model.fit(
        data_train,
        batch_size= BATCH,
        epochs=EPOCH,
        validation_data= data_test,
        verbose=1,
        callbacks= callbacks
        )

# EVALUACIÓN
results = CNN_model.evaluate(images_test, emotion_test, batch_size=128)

# GUARDADO DE LA ESTRUCTURA Y CONFIGURACIÓN EN UN .txt
file = open(SESSION_PATH + "info_model.txt", "w")
CNN_model.summary(print_fn=lambda x: file.write(x + '\n'))
file.write("\n\n" + "Learning rate: " + str(LEARNING_RATE) + "\nDecay: " +
    str(DECAY))
file.close()

# CREACIÓN DE LA MATRIZ DE CONFUSIÓN
emotions = {0:'Enfado', 1: 'Asco', 2:'Miedo', 3:'Felicidad', 4: 'Tristeza',
    5:'Sorpresa', 6:'Neutral'}
predictions = CNN_model.predict(images_test)
predictions = np.argmax(predictions, axis=1)
emotions_true=np.argmax(emotion_test, axis=1)
```

```
    conf_mat = metrics.confusion_matrix(y_true=emotions_true, y_pred=predictions)
    fig, ax = plot_confusion_matrix(conf_mat=conf_mat,
        show_normed=True,show_absolute=False, class_names=emotions.values(),
        figsize=(8,8))
    matrix_name = "MATRIZ_test_images"

    # GUARDADO DEL MODELO Y FIGURA
    accuracy = model_info.history['accuracy']
    epochs = str(len(accuracy))
    a = round(results[1], 3)
    accuracy_value = str(a)

    model_name = "model_e_"+ epochs + "-acc_" + accuracy_value +".hdf5"
    try:
        fig.savefig(SESSION_PATH + matrix_name)
        CNN_model.save(SESSION_PATH + model_name )
    except Exception:
        print("Error al guardar el modelo \n", traceback.print_exc())


if __name__ == "__main__":
    main()
```



B.3.- Programa de detección de emociones Raspberry

B.3.1.- Programa principal predicción

```
import pixel
import board
import neopixel
import keyboard

# Conexión panel LEDs
panel = neopixel.NeoPixel(board.D18, 256, auto_write=False)
color = pixel.GREY
pixel.write_text("Cargando", panel, color,0)

# Tarda unos segundos, por ello se proyecta antes un mensaje
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.python.keras.models import load_model
import os
from time import time
import _thread

### CONFIGURACIÓN ###
# Umbral de confianza de las respuestas (0-0.99)
ACCURACY = 0.10
# Se muestran las precisiones (True) de lo contrario (False)
ACC_ACTIVATED = False

# Activamos un hilo para no detener la ejecución
_thread.start_new_thread(pixel.scroll_text, ("Bienvenido", panel, 10, color))

# Archivo de modelo a utilizar
model='./modelos/modelo_CNN.hdf5'
CNN_model = tf.keras.models.load_model(model)


# Diccionario con las emociones
```

```
emotion_dict = {0: "Enfadado", 1: "Asco", 2: "Asustado", 3: "Feliz", 4: "Triste", 5:
    "Sorpresa", 6: "Neutral"}
color_dict = {0: (255, 0, 0), 1: (129, 3, 203), 2: (255, 20, 20), 3: (87,255, 0), 4:
    (69, 59, 244), 5: (244, 59, 184), 6: (59, 244, 199)}
emotion = ""
state = True

cap = cv2.VideoCapture(0)
while(state):
    ret, frame = cap.read()

    # Se notifica el comienzo de la lectura hasta captar el primer rostro
    if(emotion == ""):
        pixel.write_text('leyendo', panel, color, 0)

    # Si no hay imagen, se termina el bucle
    if not ret:
        pixel.write_text('Error', panel, color, 0)
        break
    haarcascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = haarcascade.detectMultiScale(gray, 1.3, 5)

    if keyboard.is_pressed(''):
        state=False

    for (x, y, w, h) in faces:
        roi_gray = gray[y:y + h, x:x + w]
        # Adaptacion dimensión
        image = cv2.resize(roi_gray, (48, 48))
        np_image_array = image.astype('float32') / 255.0
        img_ready = np.expand_dims(np.expand_dims(np_image_array, -1), 0)

        # Inferencia
        result = CNN_model.predict(img_ready)
        emotion_number = int(np.argmax(result))
```

```
        # Actualización panel LED
        # Se comprueba si la probabilidad es superior al filtro marcado inicialmente
        if(np.max(result) > ACCURACY):
            # Se comprueba si la emoción ha cambiado
            if(emotion_dict[emotion_number] != emotion):
                # Actualización de la emoción
                pixel.write_text(emotion_dict[emotion_number], panel,
                    color_dict[emotion_number], 0)
                emotion = emotion_dict[emotion_number]

        # Valor demasiado bajo, no puede devolverse el resultado, por tanto ?
        else:
            # Se comprueba que no se haya repetido ?
            if(emotion_dict[emotion_number] != emotion):
                # Actualización del panel
                pixel.write_text("?", panel, (255,120,84), 0)
                emotion = " ?"
                pixel.accuracy_clean(panel)
        # Se proyecta la precisión de la respuesta, si se indicó al inicio
        if(ACC_ACTIVATED):
            pixel.accuracy_print(np.max(result), panel)

cap.release()
pixel.scroll_text("Desconectando... hasta la proxima :)", panel, 10, color)
```

B.3.2.- Librería de funciones PIXEL

```
import board
import neopixel
from time import sleep
from os import urandom
from datetime import datetime

POWER_GREEN = (60, 255, 51)
GREEN = (0,5,0)
POWER_RED = (237,41,10)
RED =       (5, 0, 0)
```

```
BLUE =     (0, 0, 5)
YELLOW =   (6, 5, 0)
PINK =     (3, 1, 1)
BABY_BLUE = (0, 1, 3)
ORANGE =   (6, 2, 0)
PURPLE =   (4, 0, 6)
GREY =     (57,23,18)

COLORS = [GREEN, RED, BLUE, YELLOW, PINK, BABY_BLUE, ORANGE, PURPLE]



letters = {
    'A': ([61,62,63,64,66,125,127,128,129,130,189,191,192,194], 3),
    'B': ([61,62,63,64,66,126,127,128,129,130,189,191,192,193,194], 3),
    'C': ([61,62,63,64,127,128,191,192,193,194], 3),
    'D': ([62,63,64,66,125,127,128,130,189,191,192,193], 3),
    'E': ([61,62,63,64,127,128,129,191,192,193,194], 3),
    'F': ([61,62,63,64,126,127,128,191,192], 3),
    'G': ([61,62,63,64,127,128,130,131,191,188,192,193,194,195], 4),
    'H': ([61,63,64,66,125,126,127,128,129,130,189,191,192,194], 3),
    'I': ([61,62,63,65,126,129,190,192,193,194], 3),
    'J': ([60,61,62,66,125,130,189,191,193,194], 4),
    'K': ([60,63,64,66,126,127,128,129,189,191,192,195], 4),
    'L': ([63,64,127,128,191,192,193,194], 3),
    'M': ([59,63,64,65,67,68,123,125,127,128,132,187,191,192,196], 5),
    'N': ([60,63,64,67,124,126,127,128,130,131,188,191,192,195], 4),
    '  ': ([61,62,124,127,128,129,131,188,189,191,192,195], 4),
    'O': ([61,62,63,64,66,125,127,128,130,189,191,192,193,194], 3),
    'P': ([61,62,63,64,66,125,127,128,129,130,191,192], 3),
    'Q': ([61,62,63,64,66,125,127,128,130,189,191,192,193,194,195], 4),
    'R': ([61,62,63,64,66,125,126,127,128,129,189,191,192,194], 3),
    'S': ([61,62,63,64,126,127,129,130,189,192,193,194], 3),
    'T': ([61,62,63,65,126,129,190,193], 3),
    'U': ([61,63,64,66,125,127,128,130,189,191,192,193,194], 3),
    'V': ([61,63,64,66,125,127,128,130,189,191,193], 3),
    'W': ([59,63,64,68,123,127,128,130,132,187,189,191,193,195], 5),
    'X': ([61,63,64,66,126,129,189,191,192,194], 3),
```

```
'Y': ([61,63,64,66,125,127,129,190,193], 3),
'Z': ([61,62,63,66,125,129,191,192,193,194], 3),
'a': ([64,65,66,125,128,129,130,189,191,192,193,194], 3),
'b': ([64,127,128,129,130,189,191,192,193,194], 3),
'c': ([64,65,66,127,128,191,192,193,194], 3),
'd': ([66,125,128,129,130,189,191,192,193,194], 3),
'e': ([65,66,124,127,128,130,131,191,193,194], 4),
'f': ([64,65,66,127,128,129,191,192], 3),
'g': ([64,65,66,125,127,128,129,130,189,192,193,194], 3),
'h': ([64,127,128,129,130,189,191,192,194], 3),
'i': ([64,128,191,192], 1),
'j': ([66,130,189,191,193,194], 3),
'k': ([64,66,125,127,128,129,189,191,192,194], 3),
'l': ([64,127,128,191,192,193], 2),
'm': ([64,68,123,124,126,127,128,130,132,187,191,192,196], 5),
'n': ([64,67,124,127,128,129,131,188,189,191,192,195], 4),
'  ': ([65,66,128,129,131,188,189,191,195], 4),
'o': ([64,65,66,125,127,128,130,189,191,192,193,194], 3),
'p': ([64,65,66,125,127,128,129,130,191,192], 3),
'q': ([64,65,66,125,127,128,129,130,189,194], 3),
'r': ([64,125,126,127,128,130,191,192], 3),
's': ([64,65,66,127,128,129,130,189,192,193,194], 3),
't': ([64,65,66,126,129,190,193], 3),
'u': ([64,66,125,127,128,130,189,191,192,193,194], 3),
'v': ([64,66,125,127,128,130,189,191,193], 3),
'w': ([64,68,123,127,128,130,132,187,189,191,193,195], 5),
'x': ([64,66,125,127,129,189,191,192,194], 3),
'y': ([64,66,125,127,129,190,193], 3),
'z': ([64,65,66,125,129,191,192,193,194], 3),
' ': ([], 1),
'1': ([62,64,65,126,129,190,192,193,194], 3),
'2': ([61,62,63,66,125,129,191,192,193,194], 3),
'3': ([62,63,66,126,129,189,192,193], 3),
'4': ([61,63,64,66,125,127,128,129,130,189,194], 3),
'5': ([61,62,63,64,125,126,127,130,189,192,193,194], 3),
'6': ([61,62,63,64,127,128,129,130,189,191,192,193,194], 3),
'7': ([61,62,63,66,125,129,130,131,189,194], 3),
```

```
    '8': ([61,62,63,64,66,125,126,127,128,129,130,189,191,192,193,194], 3),
    '9': ([61,62,63,64,66,125,126,127,130,189,194], 3),
    '0': ([61,62,63,64,66,125,127,128,130,189,191,192,193,194], 3),
    '.': ([192], 1),
    ',': ([190,192], 2),
    ':': ([127,191,], 1),
    ';': ([126,190,192], 2),
    '!': ([63,64,127,128,192], 1),
    '?': ([62,64,66,125,129,193], 3),
    '  ': ([62,126,128,189,191,193], 3),
    '"': ([61,63,64,66], 3),
    "'": ([63,64], 1),
    "'": ([63,64], 1),
    '-': ([128,129,130], 3),
    '+': ([126,128,129,130,190], 3),
    '*': ([64], 1),
    '/': ([60,66,125,129,190,192], 4),
    '(': ([62,64,127,128,191,193], 2),
    ')': ([63,65,126,129,190,192], 2),
    '=': ([126,127,190,191], 2),
    '[': ([62,63,64,127,128,191,192,193], 2),
    ']': ([62,63,65,126,129,190,192,193], 2),
    '{': ([61,62,65,127,128,190,193,194], 3),
    '}': ([62,63,65,125,130,190,192,193], 3),
    '@': ([60,61,65,68,122,127,128,130,131,133,186,188,190,194,196],6),
    '$': ([61,62,63,64,126,127,129,130,189,192,193,194], 3),
    "^":([60,62,65,67,128,132,188,189,190],5),
    "  ":([60,62,65,67,129,130,131,187,191],5),
    "{":([60,62,65,67,128,132,188,189,190,194],5)
    }


def draw_letter(letter, pos, np, color):
    for point in letter:
        for r in [range(32,64), range(64,96), range(96,128), range(128,160),
            range(160,192), range(192,224)]:
            if(r.stop % 64 == 0):
```

```
                if (point in r and point - pos in r):
                    np[point - pos] = color

            else:
                if (point in r and point + pos in r):
                    np[point + pos] = color

def clean(letter, pos, np):
    color = (0,0,0)
    for point in letter:
        for r in [range(32,64), range(64,96), range(96,128), range(128,160),
            range(160,192), range(192,224)]:
            if(r.stop % 64 == 0):
                if (point in r and point - pos in r):
                    np[point - pos] = color

            else:
                if (point in r and point + pos in r):
                    np[point + pos] = color
    np.write()

def scroll_text(text, np, speed, color):
    text = list(text)
    screen = []
    while text or screen:
        if screen:
            if (screen[0]['pos'] + screen[0]['data'][1]) <= 0:
                # Comienza a eliminarse el texto por el lado izquierdo
                screen.pop(0)
            if text and ((screen[-1]['pos'] + screen[-1]['data'][1]) < 32):
                screen.append({'data': letters[text.pop(0)],
                               'pos': 32,
                               'color': color})
        else: # Se preparan las letras para proyectar
            screen.append({'data': letters[text.pop(0)],
                           'pos':32,
                           'color': color})
```

```
        for letter in screen: # Escribimos la letra en la pantalla
            draw_letter(letter['data'][0], letter['pos'], np, (170,255,100))
            letter['pos'] -= 1 # Desplazamiento
        np.write()
        np.fill([0,0,0])
        sleep(.2/speed)

def accuracy_clean(np):
    for i in range(255,223,-1):
        np[i] = (0,0,0)
    np.write()

def accuracy_print(acc, np):
    accuracy_clean(np)
    pixels_num = int(acc*32)
    pixels_num = 32- pixels_num
    if(acc >0.7):
        color = (87,230, 0)
    else:
        if (acc > 0.4):
            color = (231,76,60)
        else:
            color= (100,0,0)


    for i in range(255,223+pixels_num,-1):
        np[i] = color
    np.write()




def write_text(text, np, color,border):
    text = list(text)
    screen = []
    np.fill([0,0,0])
    np.write()
```

```
    empty = True

    while empty:
        np.fill([0,0,0])
        if screen:
            if (screen[0]['pos'] <= border):
                # Las lestras salen de la pantalla
                empty = False
            if text and ((screen[-1]['pos'] + screen[-1]['data'][1]) < 32):
                screen.append({'data': letters[text.pop(0)],
                               'pos': 32,
                               'color': color})
        else: # Se rellena la pantalla a mostrar
            screen.append({'data': letters[text.pop(0)],
                           'pos':32,
                           'color': color})
        for letter in screen: # Imprimir pantalla
            draw_letter(letter['data'][0], letter['pos'], np, color)
            letter['pos'] -= 1 # Desplazamiento

    np.write()

def digits(a):
    c = int(a%10)
    a = int(a/10)
    b = int(a%10)
    return b,c

def zero(np,color):
    draw_letter(letters[str(0)][0], 4, np, color)
    draw_letter(letters[str(0)][0], 8, np, color)
    draw_letter(letters[str(0)][0], 14, np, color)
    draw_letter(letters[str(0)][0], 18, np, color)
    draw_letter(letters[str(0)][0], 24, np, color)
    draw_letter(letters[str(0)][0], 28, np, color)
```

```
def clock(np,color):
    np.fill([0,0,0])
    h1, h2, m3, m4, s5, s6 = [0,0,0,0,0,0]
    draw_letter(letters[':'][0], 12, np, color)
    draw_letter(letters[':'][0], 22, np, color)
    zero(np,color)
    while True:
        now = datetime.now()
        d = now.hour
        d1, d2 = digits(d)

        d = now.minute
        d3, d4 = digits(d)

        d = now.second
        d5,d6 = digits(d)

        if h1!= d1:
            clean(letters[str(h1)][0], 4, np)
            h1=d1
            draw_letter(letters[str(h1)][0], 4, np, color)
            np.write()
        if h2!= d2:
            clean(letters[str(h2)][0], 8, np)
            h2=d2
            draw_letter(letters[str(h2)][0], 8, np, color)
            np.write()

        if m3!= d3:
            clean(letters[str(m3)][0], 14, np)
            m3=d3
            draw_letter(letters[str(m3)][0], 14, np, color)
            np.write()

        if m4!= d4:
            clean(letters[str(m4)][0], 18, np)
```

```
            m4=d4
            draw_letter(letters[str(m4)][0], 18, np, color)
            np.write()

        if s5!= d5:
            clean(letters[str(s5)][0], 24, np)
            s5=d5
            draw_letter(letters[str(s5)][0], 24, np, color)
            np.write()

        if s6!= d6:
            clean(letters[str(s6)][0], 28, np)
            s6=d6
            draw_letter(letters[str(s6)][0], 28, np, color)
            np.write()
```

B.3.3.- Programa detección de emociones en PC

```
import keyboard
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.python.keras.models import load_model
import os
import time
import _thread

### CONFIGURACIÓN ###
# Umbral de confianza de las respuestas (0-0.99)
ACCURACY = 0.10
# Se muestran las precisiones (True) de lo contrario (False) // Multi: mostrar una o
    varias caras
ACC_ACTIVATED = False
MULTI = True
# Archivo de modelo a utilizar
model='./modelos/modelo_CNN.hdf5'
CNN_model = tf.keras.models.load_model(model)
```

```
# Diccionario con las emociones
emotion_dict = {0: "Enfadado", 1: "Asco", 2: "Asustado", 3: "Feliz", 4: "Triste", 5:
    "Sorpresa", 6: "Neutral"}
color_dict = {0: (96, 100, 216), 1: (203, 3, 129), 2: (20, 20, 255), 3: (0,255, 87),
    4: (244, 59, 69), 5: (184, 59, 244), 6: (199, 244, 59)}
emotion_string = ""
acc=""
state = True

# FPS Variables
prev_frame_time = 0
new_frame_time = 0

# Se activa la cámara
cap = cv2.VideoCapture(0)

while(state):
    ret, frame = cap.read()

    # Si no hay imagen, se termina el bucle
    if not ret:
        print('Error, no hay imagen')
        time.sleep(2)
        break

    haarcascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = haarcascade.detectMultiScale(gray, 1.3, 5)

    for (x, y, w, h) in faces:
        new_frame_time = time.time()
        roi_gray = gray[y:y + h, x:x + w]

        # Adaptacion dimensión
        image = cv2.resize(roi_gray, (48, 48))
        np_image_array = image.astype('float32') / 255.0
```

```
    img_ready = np.expand_dims(np.expand_dims(np_image_array, -1), 0)

    # Inferencia
    result = CNN_model.predict(img_ready)
    emotion_number = int(np.argmax(result))

    if(np.max(result) > ACCURACY):
        # Actualización panel led
        if(emotion_dict[emotion_number] != emotion_string):
            emotion_string = emotion_dict[emotion_number]
    else:
        if(emotion_dict[emotion_number] != emotion_string):
            emotion_string="?"
            emotion_string = "?"

    # Cálculo de FPS
    fps = 1/(new_frame_time-prev_frame_time)
    prev_frame_time = new_frame_time
    fps = int(fps)
    color = color_dict[emotion_number]


    # Cálculo de resultados
    if(ACC_ACTIVATED):
        acc = " "+str(int(np.max(result)*100))+"%"
    if(MULTI):
        cv2.putText(frame, emotion_string+acc , (x+30,y-30),
            cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
        cv2.putText(frame, "FPS: "+str(fps) , (230,450), cv2.FONT_HERSHEY_SIMPLEX,
            1, (0,0,0), 2)
    else:
        cv2.rectangle(frame,(225,460),(590,420), (0,0,0),-1)
        cv2.putText(frame, emotion_string+acc+"FPS: "+str(fps) , (230,450),
            cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)


cv2.imshow('Video', cv2.resize(frame,(1080,720)))
```

```
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
print("Desconectando... hasta la proxima :)")
```