



**TECNOLÓGICO
NACIONAL DE MÉXICO**



Inteligencia Artificial

Reconocimiento Facial Emociones

Alumno:

Rementeria Medina Jesus Hector

Procesado de Imágenes

```
import os
import cv2
import numpy as np
from tqdm import tqdm
from sklearn.model_selection import train_test_split

# Lista de emociones que quieres detectar
CLASES = ['felicidad', 'tristeza', 'sorpresa', 'enojo']
```

Cargamos las librerías necesarias para nuestro programa y creamos la lista CLASES que contiene las emociones que el sistema va a reconocer

```
def verificar_imagen(ruta_imagen): 1 usage
    try:
        img = cv2.imread(ruta_imagen)
        if img is None:
            return False
        _ = cv2.imencode(ext='.jpg', img)[1]
        return True
    except:
        return False
```

El método verificar_imagen ayuda a detectar que imágenes de nuestro dataset pueden ser leídas correctamente y si detecta si la imagen no está corrupta

recibe la ruta donde se guardan las imágenes que vamos a utilizar y utiliza cv2 para leer las imágenes con cv2.imread y con cv2.imencode intenta codificar en memoria la imagen para detectar si no tiene ningún error y es utilizable

```

def procesar_imagenes(ruta_origen, ruta_destino, test_size=0.2): 1 usage
    for split in ['train', 'test']:
        for emocion in CLASES:
            os.makedirs(os.path.join(ruta_destino, split, emocion), exist_ok=True)

    estadisticas = {'procesadas': 0, 'errores': 0}

    for emocion in CLASES:
        print(f"\nProcesando imágenes de '{emocion}'...")
        ruta_origen_emocion = os.path.join(ruta_origen, emocion)
        ruta_train = os.path.join(ruta_destino, 'train', emocion)
        ruta_test = os.path.join(ruta_destino, 'test', emocion)

        if not os.path.exists(ruta_origen_emocion):
            print(f"Error: No se encontró el directorio {ruta_origen_emocion}")
            continue

        imagenes = [img for img in os.listdir(ruta_origen_emocion)
                     if img.lower().endswith(('.png', '.jpg', '.jpeg'))]

```

El método `procesar_imagenes` recibe la ruta donde se guardan las imágenes originales y la ruta donde se van a guardar una vez sean procesadas, así como el tamaño que se usará para hacer pruebas

Creamos un `for` que organiza las imágenes de modo que las separe en 2 conjuntos, el de entrenamiento y el de prueba y luego usando el `for emocion` las coloca según la etiqueta de emoción que le corresponde a cada foto

La variable `estadísticas` guarda cuantas de las imágenes fueron procesadas de forma correcta y cuáles fueron descartadas por contener errores

La lista `imagenes` contiene las extensiones que se tomarán en cuenta para las imágenes al momento de validarlas

```

imagenes_validas = []
print("Verificando imágenes...")
for imagen in tqdm(imagenes):
    ruta_completa = os.path.join(ruta_origen_emocion, imagen)
    if verificar_imagen(ruta_completa):
        imagenes_validas.append(imagen)
    else:
        estadisticas['errores'] += 1
        print(f"\nImagen corrupta omitida: {imagen}")

print(f"Imágenes válidas encontradas: {len(imagenes_validas)}")

train_images, test_images = train_test_split(
    *arrays: imagenes_validas, test_size=test_size, random_state=42
)

```

La lista `imagenes_validas` se encarga de ir imagen por imagen y verifica si las imágenes son válidas utilizando el método `verificar_imagen`, si una imagen contiene algún error o no puede ser utilizada correctamente aumenta la cantidad de errores en estadísticas

`train_images` divide las imágenes validadas de forma que se utilicen un 80% de estas para entrenamiento y el restante 20% para realizar pruebas

```

print("Procesando imágenes de entrenamiento...")
for imagen in tqdm(train_images):
    if procesar_y_guardar_imagen(
        os.path.join(ruta_origen_emocion, imagen),
        os.path.join(ruta_train, imagen)
    ):
        estadisticas['procesadas'] += 1

print("Procesando imágenes de prueba...")
for imagen in tqdm(test_images):
    if procesar_y_guardar_imagen(
        os.path.join(ruta_origen_emocion, imagen),
        os.path.join(ruta_test, imagen)
    ):
        estadisticas['procesadas'] += 1

print("\n Estadísticas del procesamiento:")
print(f" Imágenes procesadas exitosamente: {estadisticas['procesadas']}")
print(f" Imágenes con errores: {estadisticas['errores']}")

```

Luego se leen y procesan las imágenes usando el método `procesar_y_guardar_imagen`, esto se aplica para ambos tanto el grupo de entrenamiento como el de prueba

```
def procesar_y_guardar_imagen(ruta_origen, ruta_destino): 2 usages
    try:
        imagen = cv2.imread(ruta_origen)
        if imagen is None:
            return False


        if imagen.shape[0] < 96 or imagen.shape[1] < 96:
            print(f"\nImagen pequeña: {ruta_origen}")
            return False

        altura, ancho = imagen.shape[:2]
        if altura > ancho:
            nuevo_ancho = 96
            nuevo_alto = int(altura * (96 / ancho))
        else:
            nuevo_alto = 96
            nuevo_ancho = int(ancho * (96 / altura))

        imagen = cv2.resize(imagen, dsize: (nuevo_ancho, nuevo_alto))
        y = (nuevo_alto - 96) // 2
        x = (nuevo_ancho - 96) // 2
        imagen = imagen[y:y+96, x:x+96]
```

Este método se encarga de leer las imágenes y retocarlas, usando `image.shape` detecta si la imagen es demasiado pequeña para utilizarla si el tamaño es el correcto entonces le realiza las modificaciones, normaliza el tamaño de todas las imágenes para que contengan el mismo de 96x96 y mejora la visibilidad de la imagen

```

#Mejora de contraste
lab = cv2.cvtColor(imagen, cv2.COLOR_BGR2LAB)
l, a, b = cv2.split(lab)
clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
cl = clahe.apply(l)
 = cv2.merge((cl, a, b))
imagen = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)

cv2.imwrite(ruta_destino, imagen)
return True

except Exception as e:
    print(f"\nError procesando {ruta_origen}: {str(e)}")
    return False

```

Una vez procesadas las imágenes las envía a la ruta destino

```

if __name__ == "__main__":
    try:
        procesar_imagenes(
            ruta_origen='DataSet/originales',
            ruta_destino='DataSet/procesados'
        )
    except Exception as e:
        print(f"Error durante el procesamiento: {str(e)}")

```

En el método principal mandamos a llamar a `procesar_imagenes` y le mandamos la ruta de origen y la ruta destino para que realice todo el proceso explicado anteriormente

Cargar Datos

```
import os
import cv2
import numpy as np
from tqdm import tqdm

CLASES = ['felicidad', 'tristeza', 'sorpresa', 'enojo']
CLASES_IDX = {nombre: idx for idx, nombre in enumerate(CLASES)}

def cargar_datos(ruta_base): 3 usages
    X_train, y_train = [], []
    X_test, y_test = [], []
```

Definimos las listas CLASES con las emociones que se van a utilizar y CLASES_IDX la cual asocia cada clase con un numero

En el método cargar_datos creamos las listas X_train, Y_train y X_test e Y_test, las listas X guardan las imágenes mientras que las Y guardan las etiquetas esto según si es para entrenamiento o pruebas

```
for clase in CLASES:
    etiqueta = CLASES_IDX[clase]

    for tipo in ['train', 'test']:
        ruta = os.path.join(ruta_base, tipo, clase)
        print(f"\nCargando imágenes de {tipo} -> {clase}...")

        if os.path.exists(ruta):
            for imagen in tqdm(os.listdir(ruta)):
                try:
                    img = cv2.imread(os.path.join(ruta, imagen))
                    if img is not None and img.shape == (96, 96, 3):
                        img = img / 255.0
                        if tipo == 'train':
                            X_train.append(img)
                            y_train.append(etiqueta)
                        else:
                            X_test.append(img)
                            y_test.append(etiqueta)
                except:
                    print(f"Imagen ignorada: {imagen}")
            except Exception as e:
                print(f"Error con {imagen}: {str(e)}")
```

Este for recorre las imágenes tanto las de entrenamiento como las de prueba y obtiene su índice numero usando el diccionario de CLASES_IDX, las imágenes son leídas con cv2.imread y verifica si se cumple con las dimensiones esperadas y que contenga los 3 canales de RGB, si las imágenes son válidas las normaliza dividiendo sus pixeles en 255 convirtiendolos dentro de un rango 0 a 1 lo cual se utiliza para estabilizar el entrenamiento de la red neuronal

Dependiendo de si la imagen es entrenamiento o prueba se agrega a su correspondiente lista x_train o x_test con su respectiva etiqueta en y_train o y_test, esto asegura que todos los datos estén correctamente estructurados y listos para ser convertidos en arreglos de NumPy para alimentar el modelo

```
X_train, X_test = np.array(X_train), np.array(X_test)
y_train, y_test = np.array(y_train), np.array(y_test)

print("\nFormas finales:")
print(f"X_train: {X_train.shape}, y_train: {y_train.shape}")
print(f"X_test: {X_test.shape}, y_test: {y_test.shape}")

return X_train, y_train, X_test, y_test

if __name__ == "__main__":
    cargar_datos('DataSet/procesados')
```


Modelo

```
import tensorflow as tf

def crear_modelo(input_shape=(96, 96, 3), num_clases=4): 3 usages
    entradas = tf.keras.Input(shape=input_shape)

    x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu')(entradas)
    x = tf.keras.layers.MaxPooling2D(2, 2)(x)

    x = tf.keras.layers.Conv2D(64, (3, 3), activation='relu')(x)
    x = tf.keras.layers.MaxPooling2D(2, 2)(x)

    x = tf.keras.layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = tf.keras.layers.MaxPooling2D(2, 2)(x)

    x = tf.keras.layers.Flatten()(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    salidas = tf.keras.layers.Dense(num_clases, activation='softmax')(x)

    modelo = tf.keras.Model(inputs=entradas, outputs=salidas)
```

Definimos como será nuestro modelo base, le decimos que utilizara de entrada imágenes con formato 96x96 con 3 canales los cuales son el RGB y usamos num_clases para definir la cantidad de etiquetas que utilizara el modelo en este caso serán 4 que pertenecen a las 4 emociones que vamos a utilizar

Luego se definen las capas que usara el modelo, en este caso utilizara 3 capas convolucionales, una capa intermedia y una de salida

```

    modelo.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    return modelo

✓ if __name__ == "__main__":
    modelo = crear_modelo()
    modelo.summary()

```

Compilamos el modelo colocándole una tasa de aprendizaje inicial de 0.001 y con metrics se evalúa el rendimiento del modelo usando la precisión

Entrenamiento

```

def entrenar_modelo(X_train, y_train, X_test, y_test): 1 usage
    modelo = crear_modelo()

    os.makedirs(name='detector_emociones/modelos', exist_ok=True)

    callbacks = [
        tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True),
        tf.keras.callbacks.ModelCheckpoint(
            'detector_emociones/modelos/mejor_modelo.keras',
            monitor='val_accuracy',
            save_best_only=True
        ),
        tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3, min_lr=1e-6)
    ]

    historia = modelo.fit(
        X_train, y_train,
        epochs=50,
        batch_size=32,
        validation_data=(X_test, y_test),
        callbacks=callbacks,
        verbose=1
    )

```

El método `entrenar_modelo` recibe de entrada las listas X e Y de entrenamiento y prueba

Creamos el directorio donde se va a guardar el modelo una vez sea entrenado

Configuramos callbacks para que lleve a cabo el control del entrenamiento, callbacks detendrá el entrenamiento si detecta que dejó de mejorar para así ahorrar tiempo, también guarda automáticamente el mejor modelo posible durante el entrenamiento y reduce la velocidad de aprendizaje si el modelo se estanca

Con `historia` llevamos a cabo el entrenamiento del modelo mostrándole las imágenes junto con sus etiquetas para que aprenda

```
def evaluar_modelo(modelo, X_test, y_test): 1 usage
    resultados = modelo.evaluate(X_test, y_test, verbose=1)
    print("\nResultados:")
    for nombre, valor in zip(modelo.metrics_names, resultados):
        print(f"{nombre}: {valor:.4f}")

if __name__ == "__main__":
    try:
        X_train, y_train, X_test, y_test = cargar_datos('DataSet/procesados')
        modelo, historia = entrenar_modelo(X_train, y_train, X_test, y_test)
        evaluar_modelo(modelo, X_test, y_test)
    except Exception as e:
        print(f"Error durante entrenamiento: {str(e)}")
```

El método `evaluar_modelo` recibe el modelo entrenado y las imágenes que usaremos para realizar las pruebas

Usando `evaluate` el modelo realiza pruebas con las imágenes que reservamos y mide el rendimiento del modelo

```
108/108 ————— 15s 138ms/step - accuracy: 0.9507 - loss: 0.1569 -  
Epoch 8/50  
108/108 ————— 15s 140ms/step - accuracy: 0.9695 - loss: 0.0941 -  
Epoch 9/50  
108/108 ————— 14s 131ms/step - accuracy: 0.9728 - loss: 0.0708 -  
Epoch 10/50  
108/108 ————— 16s 146ms/step - accuracy: 0.9824 - loss: 0.0485 -  
Epoch 11/50  
108/108 ————— 22s 159ms/step - accuracy: 0.9928 - loss: 0.0256 -  
27/27 ————— 1s 28ms/step - accuracy: 0.8427 - loss: 0.4836
```

```
Resultados:|  
loss: 0.5152  
compile_metrics: 0.8384
```

Reconocimiento Cámara

```
import cv2  
import numpy as np  
import tensorflow as tf  
  
CLASES = ['Felicidad', 'Tristeza', 'Sorpresa', 'Enojo']  
|  
def cargar_modelo(ruta_modelo='detector_emociones/modelos/mejor_modelo.keras'):  
    return tf.keras.models.load_model(ruta_modelo)
```

Creamos la lista CLASES con las emociones y cargamos el modelo ya entrenado

```
def deteccion_tiempo_real(): 1 usage  
    print("Cargando modelo...")  
    modelo = cargar_modelo()  
  
    detector_facial = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')  
    cap = cv2.VideoCapture(0)  
  
    if not cap.isOpened():  
        print("No se pudo abrir la cámara")  
        return  
  
    print("Presiona 'esc' para salir")
```

Detector_facial carga un clasificador Haar de la librería OpenCV el cual ya está pre entrenado para detectar caras, el detector ya viene con una interfaz de cámara integrada

cap.isOpened se encarga de detectar si la cámara está funcionando de forma correcta, de lo contrario detiene la función

```
while True:|
    ret, frame = cap.read()
    if not ret:
        break
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    rostros = detector_facial.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
```

Con el While mantenemos el programa funcionando constantemente hasta que el usuario decida cerrarlo, usamos ret para que la cámara en tiempo real lea cada fotograma, si no puede capturar un fotograma se sale del bucle al detectar un error

gray se encarga de convertir la imagen a escalas de blanco y negro para facilitarle al sistema el detectar las caras

rostros se encarga de leer esta imagen y detectar los rostros presentes en el fotograma actual

```
for (x, y, w, h) in rostros:
    rostro = frame[y:y+h, x:x+w]
    try:
        rostro_procesado = cv2.resize(rostro, dsize=(96, 96))
        rostro_procesado = rostro_procesado / 255.0
        rostro_procesado = np.expand_dims(rostro_procesado, axis=0)

        predicciones = modelo.predict(rostro_procesado, verbose=0)
        clase_idx = np.argmax(predicciones[0])
        emocion = CLASES[clase_idx]
        confianza = predicciones[0][clase_idx]
```

Este for se encarga de procesar cada rostro detectado, luego lo prepara para ser usado dentro del modelo, redimensionándolo a 96x96 pixeles y normalizando los valores

Luego el modelo realiza una predicción calculando la probabilidad para cada emoción y de entre todas elige la emoción que tuvo el mayor porcentaje y usando `clase_idx` la convierte según el número que le corresponde en una etiqueta textual

```
color = (0, 255, 0)
texto = f"{emocion} ({confianza*100:.1f}%)"
cv2.rectangle(frame, (x, y), (x+w, y+h), color, 2)
cv2.putText(frame, texto, org=(x, y-10), cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.9, color, thickness=2)
```

Una vez el modelo decide cual es la emoción detecta en el rostro este dibuja alrededor de la cámara un recuadro junto con un texto el cual dice cuál es la emoción que se detectó junto con su porcentaje de fiabilidad

```
cv2.imshow(winname='Detector de Emociones', frame)

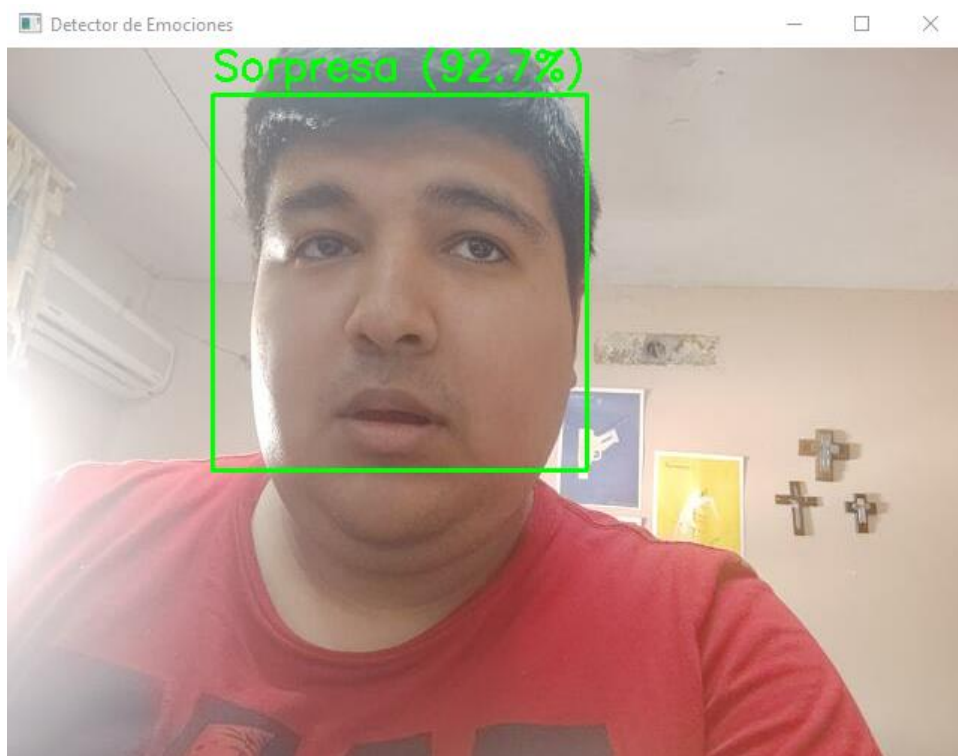
if cv2.waitKey(1) & 0xFF == 27:
    break

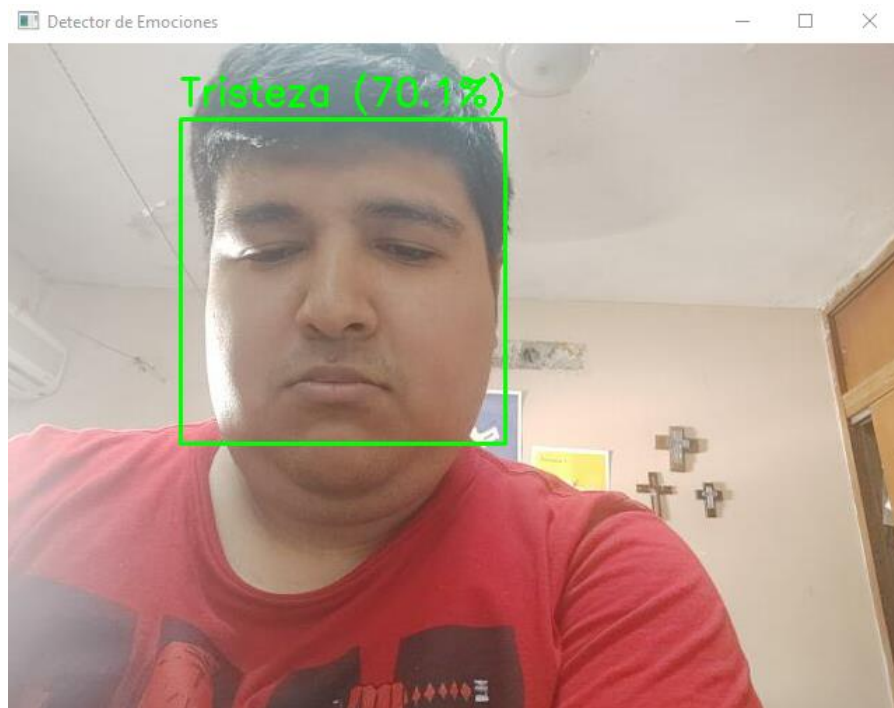
cap.release()
cv2.destroyAllWindows()

if __name__ == "__main__":
    try:
        deteccion_tiempo_real()
    except Exception as e:
        print(f"Error: {str(e)}")
```

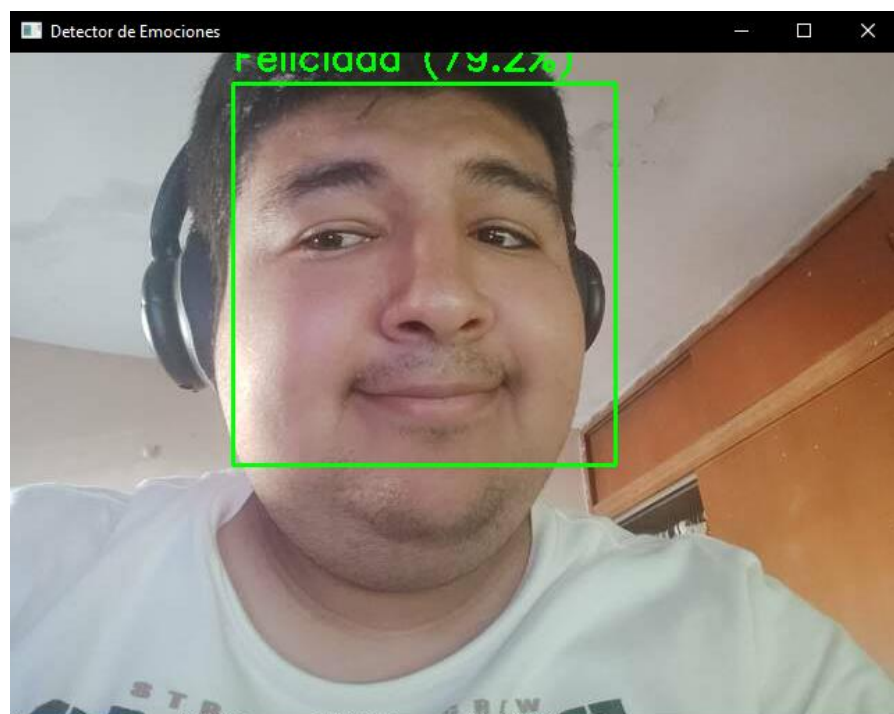
Por ultimo `waitKey` se encarga de detectar si el usuario pulso la tecla `esc` para cerrar la ventana y teminar el porgrama usamos `cap.release` y `destroyAllWindows` para liberar recursos cerrando la cámara y la ventana de OpenCV

Ejemplos:

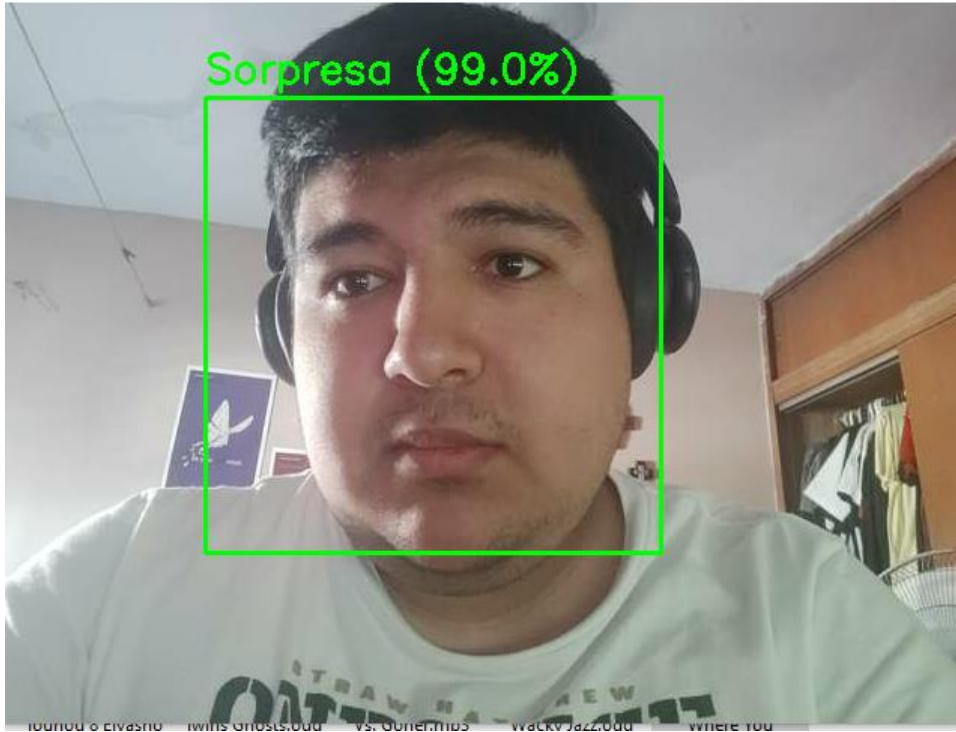




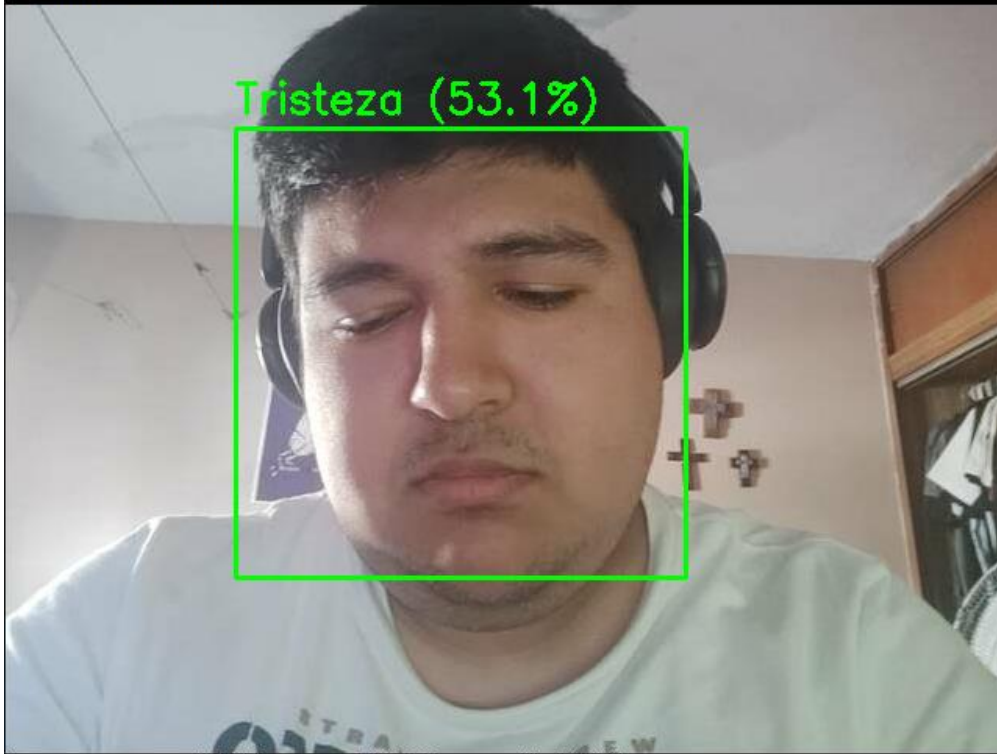
Ejemplos con baja iluminación:



Sorpresa (99.0%)



Tristeza (53.1%)

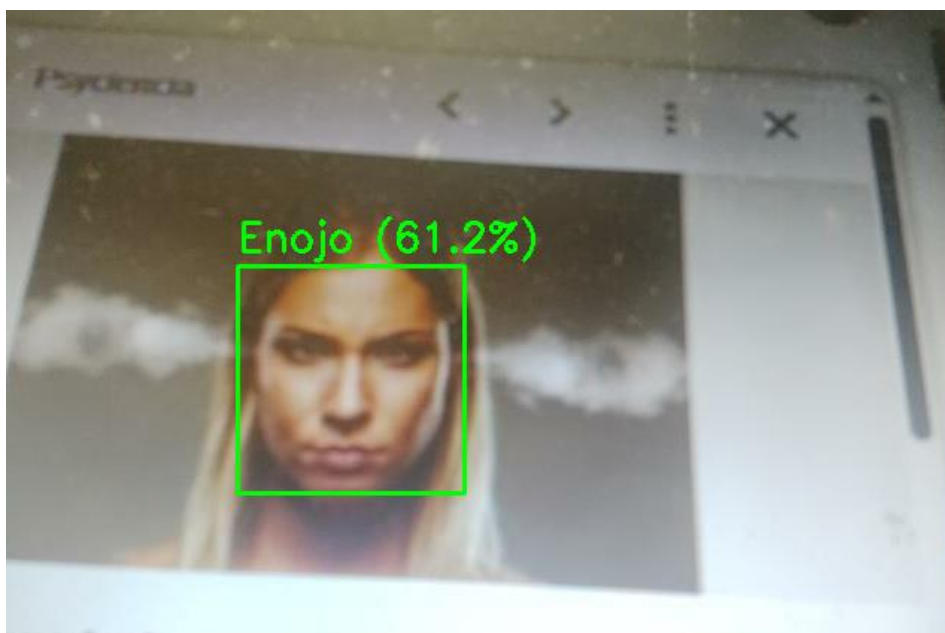
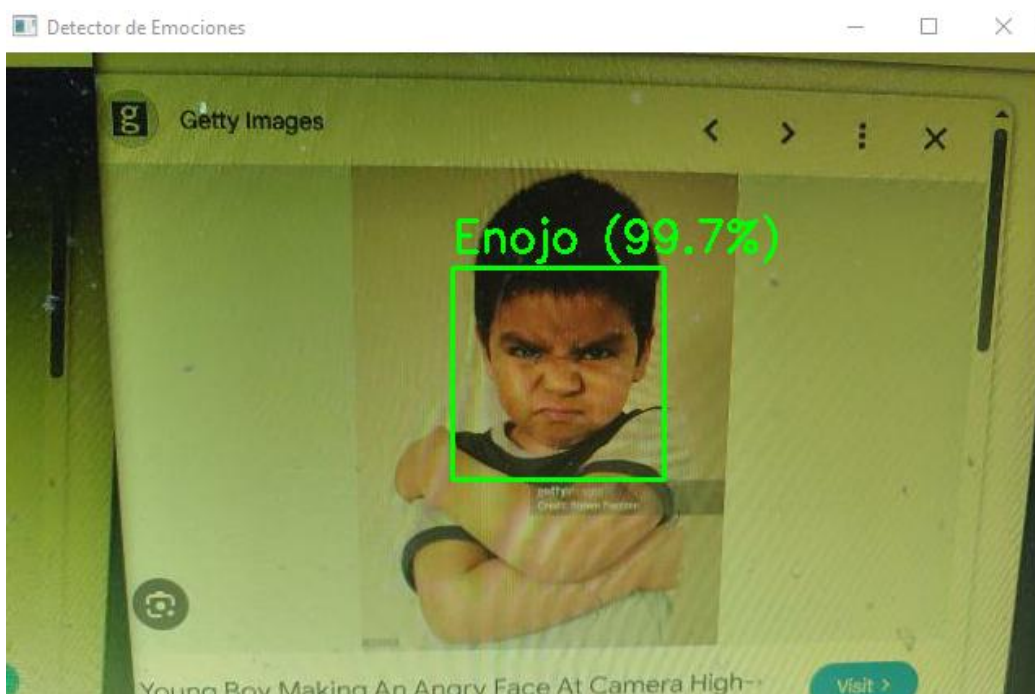


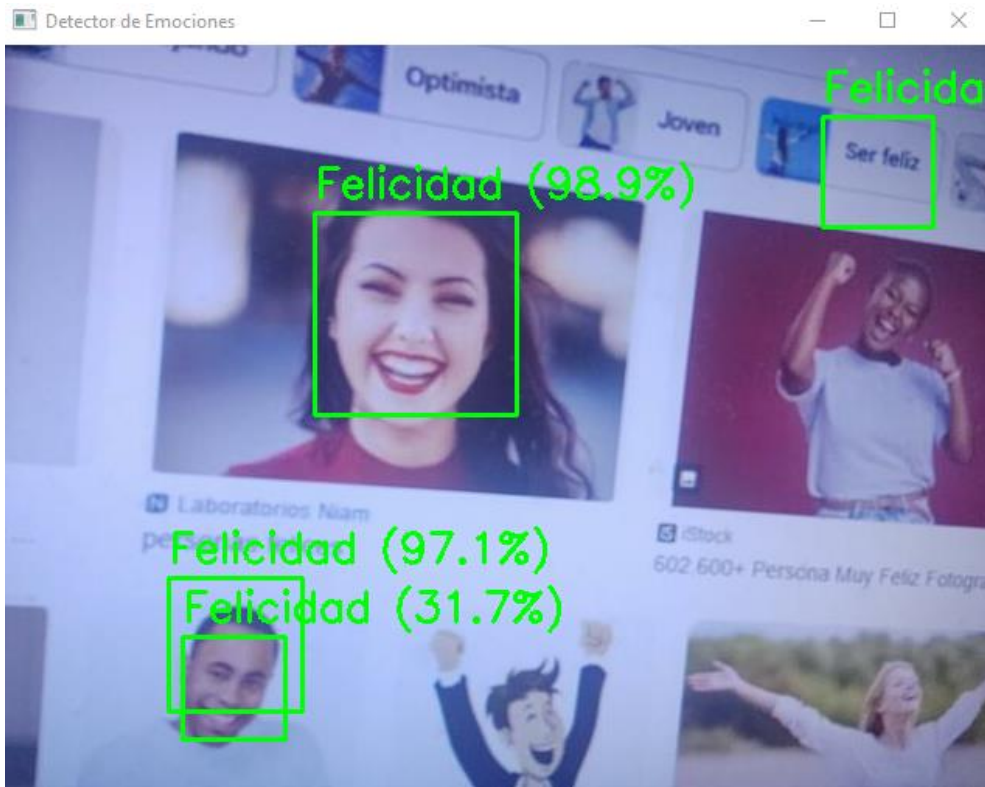
Video Demostración:

<https://www.youtube.com/watch?v=-5TRtH27gfo>

No logre hacer una cara de enojo que el modelo detectara como válida, pero haciendo pruebas con imágenes aleatorias sacadas de google si logro detectar cuando una imagen era de una persona enojada

Ejemplos usando imágenes de google:





Este ejemplo es para demostrar que también detecta las demás emociones con imágenes