INSTITUTO TECNOLÓGICO DE CULIACÁN

INGENIERÍA EN SISTEMAS COMPUTACIONALES

ASIGNATURA:

INTELIGENCIA ARTIFICIAL

DOCENTE:

ZURIEL DATHAN MORA FELIX

ALUMNOS:

URIAS ESCOBAR JOSE MANUEL

FLORES BELTRAN JESUS GUADALUPE

ASIGNACION:

PROYECTO MODULO IV: APLICACIÓN DE DETECCION FACIAL

FECHA DE ENTREGA:

29/11/2025

MUESTRAS DEL FUNCIONAMIENTO DE EL PROGRAMA (VIDEO):

Prueba de funcionamiento del programa (INTELIGENCIA ARTIFICIAL)

evidencia

INTRODUCCION

Para el desarrollo de este proyecto, se realizaron varias pruebas con diferentes ideas de desarrollo de proyecto, intentamos inicialmente crear la arquitectura de la red neuronal desde 0 y entrenarla con un dataset como FER2013 y con datasets creados por nosotros, pero resulto ser mas complicado de lo que parece, el entrenamiento de la red neuronal resulto ser muy ineficiente, pues nunca nos dio los resultados deseados, no es cosa sencilla hacer que nos de resultados casi perfectos en cámara en vivo, si incluso con imágenes nos daba errores mucho menos nos daría resultados correctos en vivo.

Entonces evaluamos las alternativas que venían en las instrucciones del proyecto y decidimos utilizar un modelo ya preentrenado para el desarrollo de este proyecto.

Ejemplos como VGG16 o ResNet. Investigando encontramos un modelo llamado Emotion-FERPlus basado en CNN es de tipo ResNet, me pareció interesante pues en teoría hasta lo que conozco de este modelo no requiere entrenar nada.

Asi que Manos a la obra.

LIBRERIAS A INSTALAR

1. Pip install onnxruntime: para cargar y ejecutar el modelo.
2. Pip install opencv-python: para manejo de cámara y mostrar video.
3. Pip install numpy: para procesar matrices e imágenes.

DESCARGAR EL MODELO PREENTRENADO

Modelo entrenado con el dataset FER2013, mejorado con etiquetas mas precisas bajo el proyecto FER+.

- 35,000 imágenes de rostros reales.
- 8 emociones etiquetadas manualmente.
- Expresiones exageradas y controladas.

Descarguamos el modelo del siguiente repositorio de github:
models/validated/vision/body_analysis/emotion_ferplus/model/emotion-ferplus-8.onnx at main · onnx/models

De la documentación oficial nos habla de:

La entrada del modelo: tensor de forma (1, 1, 64, 64)

- 1: batch.
- 1: canal (escala de grises).
- 64x64: tamaño de la imagen.

Salida: vector de forma (1, 8) con clases para 8 emociones.

Tabla_emociones = {

0: ‘neutral’,

1: ‘happiness’,

2: ‘surprise’,

3: ‘sadness’,

4: ‘anger’,

5: ‘disgust’,

6: ‘fear’,

7: ‘contempt’

}

Entonces se tiene que pasar la cara al modelo en escala de grises, redimensionar a 64x64, darle forma (1,1,64,64) y aplicar un softmax a los 8 puntuaciones para obtener las probabilidades.

DETALLANDO UN POCO COMO FUNCIONA EL MODELO

El modelo Emotion Recognition Plus, es una red neuronal convolucional (CNN) preentrenada desarrollada principalmente por Microsoft Research para reconocer expresiones humanas a partir de imágenes de rostros. Este modelo se distribuye en formato ONNX, entonces se permite ejecutar con librerías como ONNX Runtime de forma rápida e incluso usarla sin **GPU**.

Arquitecuta general del modelo:

Esta basado en una Red Neuronal Convolucional (CNN).

Una CNN esta compuesta por varias capas diseñadas para procesar imágenes:

- **Capas convolucionales:**
  - detectan patrones visuales como: bordes, líneas, sombras.
  - Luego patrones mas complejos como cejas fruncidas, apertura de ojos, forma de boca, entre otros.
- **Capas de activación (ReLu):**
  Introducen la no linealidad para que el modelo pueda aprender patrones complejos.
- **Capas de pooling (MaxPooling):**
  - Reducen el tamaño de la imagen, manteniendo los rasgos mas importantes del rostro.
- **Capas densas:**
  - Clasifican las características extraídas en una de las emociones posibles.
- **Capa softmax:**
  - Convierte los resultados numéricos en probabilidades (0.0-1.0)

Entonces el modelo tiene las siguientes clases con las que trabaja:

Numero	Emocion
0	Neutral
1	Happiness
2	Surprise
3	Sadness
4	Anger
5	disgust
6	fear
7	contempt

El flujo interno del modelo funciona de la siguiente forma:

Entrada que tiene el modelo:

La red espera una imagen 64x64 pixeles, en escala de grises, representado así -> (1, 1, 64, 64).

Obtencion de características:

Las primeras capas (CNN) identifican patrones visuales como las siguientes:

- Posición de cejas.
- Tensión de los parpados.
- Apertura de ojos.
- Arrugas en la frente.
- Manos en la parte boca para detectar tristeza.

Esos son algunas características que se transforman en un conjunto de mapas de activación.

Dependiendo de nuestros gestos con el rostro, a partir de la forma en como estuvo construido el dataset para el entrenamiento de este modelo podrá aprender mediante el procesamiento visual las relaciones que tiene el rostro con las emociones.

Obtuve el dataset FER2013 para conocer un poco como este modelo puede detectar las emociones, y estos son unos ejemplos:

Tristeza:

Enojo:

Neutral:

Felicidad:

Al final, la capa softmax produce 8 valores numéricos que suman 1 (100%)

La emoción con mayor probabilidad es la predicción final.

ARQUITECTURA DE NUESTRO SISTEMA

El sistema estará compuesto por dos módulos principales:

1. Uno de detección de rostro (OpenCV – HaarCascade): se encarga de identificar la región donde aparece una cara dentro del frame del video. Entonces esto nos servirá para que nos regrese las coordenadas de donde esta específicamente el rostro.
2. Modulo de reconocimiento emocional (Modelo FER+ ONNX): una CNN preentrenada que reciba nuestro rostro recortado, lo procesa y produce un vector con probabilidades para las emociones. A partir de ese vector se seleccionan las más probables.

Elegimos OpenCV pues nos permite abrir la cámara en tiempo real, convertir imágenes al formato que pide el modelo, dibujar resultados en pantalla e integrar el modelo. Además, es liviano para la laptop que tenemos.

Ademas Elegimos HaarCascade para detectar rostros, aunque existen alternativas modernas como (MediaPipe, YOLO-face), Haarcascade nos pareció ligero, fácil de integrar, funciona bien para lo que queremos.

Haarcascade lo descargamos del siguiente repositorio de github:
opencv/data/haarcascades/haarcascade_frontalface_default.xml at master · opencv/opencv

HAARCASCADE

Es un detector preentrenado de detección de rostros, creado por Paul Viola y Michel Jones. Fue uno de los primeros algoritmos que permitieron detectar caras rápidamente en tiempo real, incluso en computadoras lentas como la de nosotros.

El algoritmo usa:

- **Características Haar: que son patrones visuales simples (bordes, líneas, sombras).**
- **Clasificadores en cascada: que son una cadena de filtros que descartan zonas que no son rostros.**

- **Ventanas deslizantes: barrido de toda la imagen buscando patrones similares a una cara.**

CONOCIENDO OPENCV

Para comenzar a trabajar, primero estuve viendo un poco como usar OpenCV y que muestre video de la webcam a la pc con el programa, para ello importe:

- Import cv2: que es la librería de OpenCV, sirve para manejar video, imágenes, manejar la cámara mas que nada.

Para abrir la cámara uso el siguiente código:

Cámara= cv2.videoCapture(1, cv2.CAP_DSHOW)

- **VideoCapture(1):** significa que estoy usando la cámara numero 1. Pues la cámara 0 que es de la laptop esta dañada en mi caso).
- **cv2.CAP_DSHOW**: le dice a Windows que use el backend "DirectShow" que es mas compatible para webcams externas(RECOMENDADO).

Crea el objeto que me permite leer frames.

Este bucle es esencial, pues aquí se lee continuamente la cámara, mustra el frame en pantalla.

While True:

resultado, video_de_camara = cámara.read() -> .read()

resultado: dice si la lectura fue exitosa o no (true o false), si es falso no obtiene nada y sale del bucle

video_de_camara: es la imagen capturada.

- **Cv2.imshow**("nombre de la ventana"): muestra la imagen de la cámara en una ventana.
- **Cv2.waitkey(1)** & 0xFF == ord('s'): esta parte detecta si se presiona la letra e.
    - Waitkey(1): espera 1 ms por una tecla.
    - & 0xFF : limpia bits extra (obligatorio para opencv).
    - Ord('s'): convierte la letra en su código ASCII.

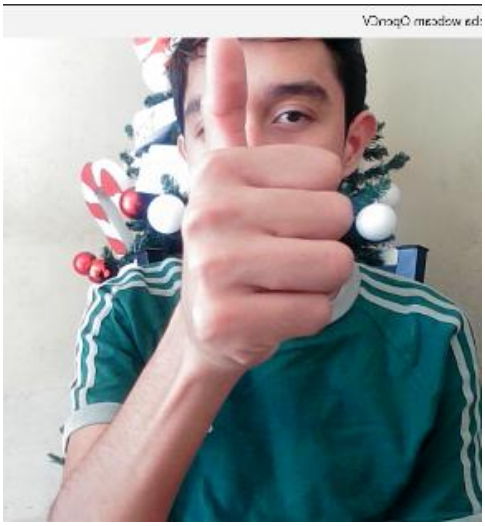
Camara.release():

libera la camara para que otras apps la puedan usar, si no se cierra correctamente la camara, puede quedar como ocupada.

Cv2.destroyAllwindows():

cierra todas las ventanas que se abrieron con OpenCV.

Imagen donde muestro el uso de OpenCV.



CONOCIENDO EL USO DEL MODELO CON FOTOS.

Esto porque quiero ver como es que se manda la imagen a el modelo y que me de resultados.

Librerías usadas:

Cv2: para manejar imágenes (leerlas, convertir a grises, redimensionamiento)

Onnxruntime: para cargar y ejecutar el modelo en formato ONNX

Numpy: para manejo de arreglos numéricos (hacer softmax, buscar los máximos).

 # lista de emociones en el orden que usa el modelo FER+

emociones = [

"neutral",

"feliz",

"sorpresa",

"triste",

"enojo",

"asco",

"miedo",

"desprecio"

]

Esto lo hago para cuando el modelo indique el índice que ya tiene preentrenado yo lo traduzco a el resultado usando esta lista.

Carga de el modelo:

- **Modelo_de_emociones= ort.inferencesession(“emotion-ferplus-8.onnx”)**
  - Abre el archivo .onnx.
  - Carga la red neuronal preentrenada.
- **InferenceSession** es un objeto que representa el modelo cargado. Con eso ya se puede mandar imágenes al modelo y recibir predicciones.

Lectura de la imagen:

Cv2.imread:

- **Carga una imagen desde el directorio.**
- Ejemplo:
- Imagen = cv2.imread(“ejemplo.jpg”)

Cv2.cvtColor (imagen, cv2.COLOR_BGR2GRAY), se uso **cvtColor** porque:

- **Convierte a la imagen de color a escala de grises. (cv2.COLOR_BGR2GRAY)**
- **Pues el modelo FER+ fue entrenado con imágenes en blanco y negro no a color.**

Rostro_tamaño_64x64= Cv2.resize(gris, (64,64)), su uso para:

- Cambiar el tamaño de la imagen gris a 64x64px.
- Pues es ese el tamaño de entrada que el modelo espera.

Rostro_tamaño_64x64.astype("float32).reshape(1, 1, 64, 64):

.astype("float32):

Convierte los valores de la imagen a tipo float32, porque? Porque opencv lee las imágenes normalmente como unit8 (0 a 255)

Este modelo trabaja con float32.

Con astype("float32") convierto la imagen al tipo numérico que espera el modelo.

.reshape(1, 1, 64, 64):

Cambia la forma del arreglo, el modelo espera (1, 1, 64, 64), como lo explique al inicio:

La entrada del modelo: tensor de forma (1, 1, 64, 64)

- 1: batch.
- 1: canal (escala de grises).
- 64x64: tamaño de la imagen.

Input_name= modelo.get_inputs()[0].name y output_name = modelo.get_outputs()[0].name:

Esto lo utilizo porque leyendo, dice, que muchos modelos ONNX tienen nombres internos para sus entradas y salidas.

- Cont **get_inputs()[0].name** obtienes el valor de la primera entrada.
- Con **get_outputs()[0].name** obtienes el nombre de la salida.

Todo esto para saber por donde le mando la imagen y por donde me tiene que regresar el resultado.

Modelo_de_emociones.run([modelo.output_name], {modelo.input_name: entrada}):

Aquí es donde ya se ejecuta el modelo, el primer output_name indico que salida quiero calcular, y input_name manda los datos de la entrada.

Calculo de resultados:

probabilidad_para_la_emocion = Resultados(puntuaciones_de_emociones)[0]

porcentaje = np.exp(probabilidad_para_la_emocion - np.max(probabilidad_para_la_emocion)): calcula las exponenciales de los valores que el modelo da, pero le resto el valor máximo para evitar números grandes.

probabilidades= porcentaje / np.sum(porcentaje): esta función lo que hace es convertir valores a probabilidades dividiendo los exponenciales entre la suma total. Esto genera probabilidades que suman 1.

probabilidad_para_la_emocion = Resultados(puntuaciones_de_emociones)[0]: el modelo devuelve un arreglo de un tamaño de longitud 8 (1,8), con probabilidad[0] obtengo la fila y se queda con un arreglo de 8 elementos, cada uno es una probabilidad.

Np.argmax(probabilidad_para_la_emocion): esto hace que nos devuelva el índice mayor en probabilidad, si el máximo esta en por ejemplo la posición 1 nos regresara feliz, si es 0 neutro y así.

Y ese índice lo uso para mi lista de nombres de emociones.

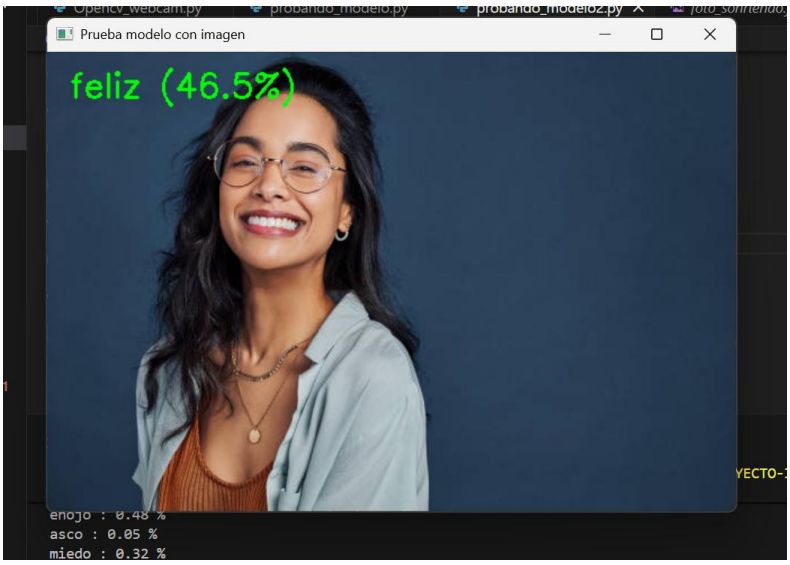
Ejemplo de uso:

- indice_mas_alto_probable= int(np.argmax(probabilidad_para_la_emocion))
- emocion_resultante= Emociones[indice_mas_alto_probable]

impresión de resultado en la imagen:

```
For i in range (len(probabilidad_para_la_emocion)):
    Print(Emociones[i], ":", round(probabilidad_para_la_emocion[i]*100,2),
    "%")
```

Hice una pequeña prueba y me dio el siguiente resultado, que me pareció bastante curioso:



CONOCIENDO HAARCASCADE FRONTALFACE CON OPENCV

Quiero ver como funciona la detección de rostros con Haarcascade.

Para ello primero cargue el archivo de la siguiente forma:

haarcascade = cv2.CascadeClassifier(ruta_del_haar_cascade), crea un objeto detector de rostro utilizando un archivo xml.

Después se requiere obtener el rostro detectado, eso lo hago de la siguiente forma, todo dentro del while donde valido si la cámara esta activa:

Despues aplico los mismos filtros de escala de grises, redimensionamiento de imágenes.

Aplico lo siguiente para detectar el rostro

```
coordenadas_de_el_rostro = haarcascade.detectMultiScale(
    video_a_escala_de_grises,
    scaleFactor=1.3,   reduce el tamaño de la imagen para la escala de el rostro
    minNeighbors=5,  control de detecciones para aceptar una cara
    minSize=(60, 60)    tamaño del rostro
)
```

Indico si detecta al menos 1 rostro en el siguiente if:

```
# le digo que si detecta al menos un rostro
if len(coordenadas_de_el_rostro) > 0:
x, y, w, h = coordenadas_de_el_rostro[0]
```

después para que en cámara se vea la detección de la cara muestro un recuadro, con el siguiente código:

cv2.rectangle(

- **video_de_camara,** lugar donde se mostrara el rectangulo
- **(x,y),** x: posición horizontal, y posición vertical.
- **(x + w, y + h),** w es el ancho del rostro, h es lo alto del rostro
  x + w = se mueve desde la x inicial hasta el final de la cara
  y + h = se mueve desde la y inicial hasta debajo de la cara
- **(0, 255, 0),** color del rectángulo en BGR, que admite opencv, en este caso es verde
- **2** es el grosor de la linea

)

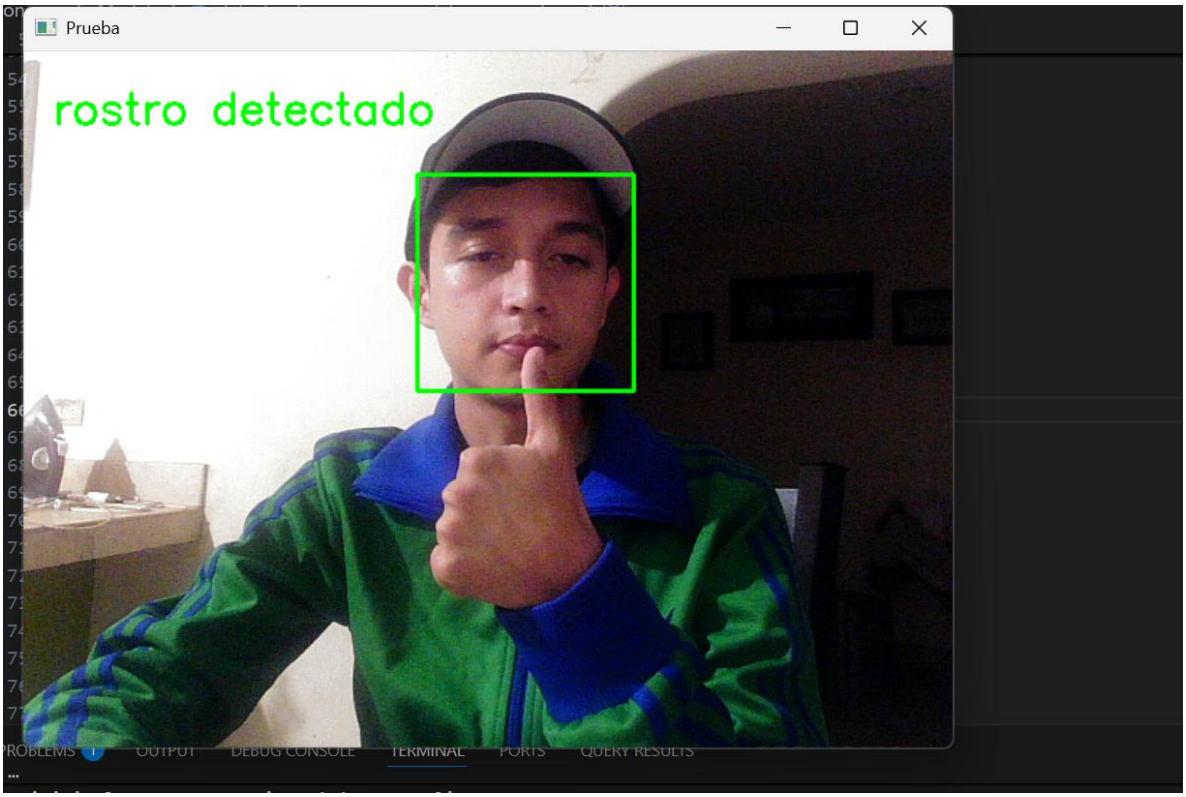
Luego para escribir un texto para decir que se detecto el rostro con el siguiente código:

cv2.putText(

- **video_de_camara,** imagen donde se escribira el texto
- **texto_a_mostrar_pantalla,** texto que se mostrara en pantalla
- **(20, 50), 20:** posición en x del inicio del texto, 50 posicion en y
- **cv2.FONT_HERSHEY_SIMPLEX,** tipo de fuento del texto
- **1.0,** tamaño de la fuente
- **(0, 255, 0),** color de la fuente
- **2** grosor del texto

)

Muestra de funcionamiento



DETECTOR DE EMOCIONES

Para poder llegar a desarrollar el detector de emociones, realice pruebas, como anteriormente mostramos, utilización de OpenCV para al menos ver video, después implementación de el modelo preentrenado con una imagen para ver que resultados daba, y como se vio anteriormente utilización de Haarcascade para detección de rostro.

Por ultimo toco implementar el Haarcascade junto con el modelo preentrenado para que nos diera resultados con la cámara en tiempo real.

Para ello utilice 3 archivos Python los cuales llame:

- **Detector_de_rostro.py**
- **Detector_de_webcam.py**
- **Modelo_emociones.py**

Lo dividimos de esa forma para tener todo bien organizado y no perdernos con tanto cochinero en el código.

Se integro Haarcascade con Emotion-ferplus-8 porque el modelo puede analizar las emociones si se recibe un rostro limpio y centrado.

En el archivo detector_de_rostro:

Se encarga de detectar los rostros de la cámara, utilizando el haarcascade, convierte el video a escala de grises, y extrae las coordenadas del primer rostro detectado.

De la siguiente forma:

```
coordenadas_de_el_rostro= coordenadas_de_el_rostro[0]

    x = coordenadas_de_el_rostro[0]

    y = coordenadas_de_el_rostro[1]

    w = coordenadas_de_el_rostro[2]

    h = coordenadas_de_el_rostro[3]

    return (x, y, w, h)
```

el archivo Modelo_emociones.py: contiene todo lo relacionado con el modelo preentenado.

- **Se carga el modelo**
- **Redimensiona a 64x64**
- **Convierte a float32**
- **da forma a (1, 1, 64, 64)**

Ejecucion del modelo, se aplican cálculos para obtener resultados de la probabilidad de la emoción, como se explico anteriormente.

Y por ultimo, la clase principal:

Detector_webcam.py

Este es el que une todo, se encarga de la webcam, bucle del video, detección de el rostro, el recorte de la parte donde esta el rostro, predicción de el modelo y obviamente la visualización en cámara.

Se definen aquí las rutas de el modelo preentrenado y de el Haarcascade

- **ruta_del_modelo= "emotion-ferplus-8.onnx"**
- **ruta_del_haar_cascade = "haarcascade_frontalface_default.xml"**

Hace uso de las clases anteriormente mencionadas así:

- **modelo_de_emociones = ModeloDeEmociones(ruta_del_modelo)**
- **detector_de_rostros = DetectorDeRostro(ruta_del_haar_cascade)**

para hacer el recorte de el rostro hace lo siguiente:

rostro_extraido = video_de_camara[y:y+h, x:x+w]: y: y+h, desde la fila y hasta la fila Y+h, O sea desde el inicio vertical del rostro hasta su final, x: x+w desde la columna x hasta la columna x+w o sea desde el inicio horizontal hasta su final.

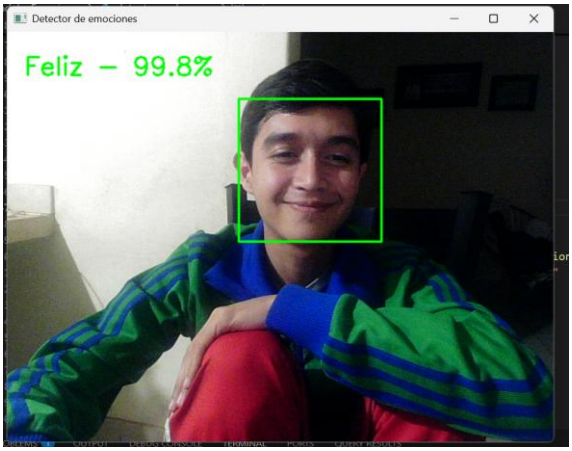
Después para que el modelo pueda decir que emoción es la que la persona esta mostrando en pantalla se hace lo siguiente:

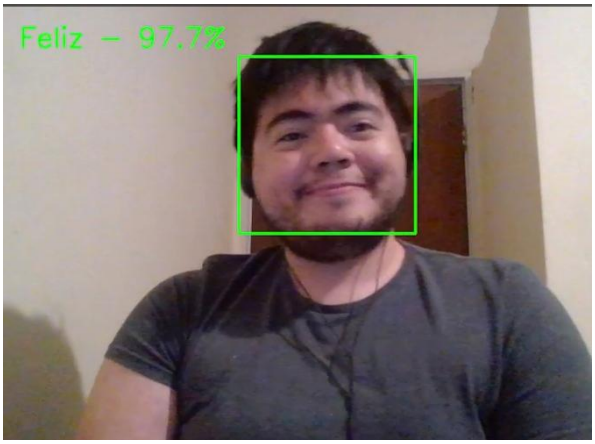
Se manda el rostro extraído al modelo de emociones, para obtener la emoción y la confianza, después se imprime en pantalla la emoción resultante y la el porcentaje de confianza.

```
try:
        emocion, confianza =
        modelo_de_emociones.Reconocimiento_de_emocion(rostro_extraido)

        texto_a_mostrar_pantalla = f"{emocion} - {round(confianza, 1)}%"

except Exception as e:

        texto_a_mostrar_pantalla = "error en predicción"
```

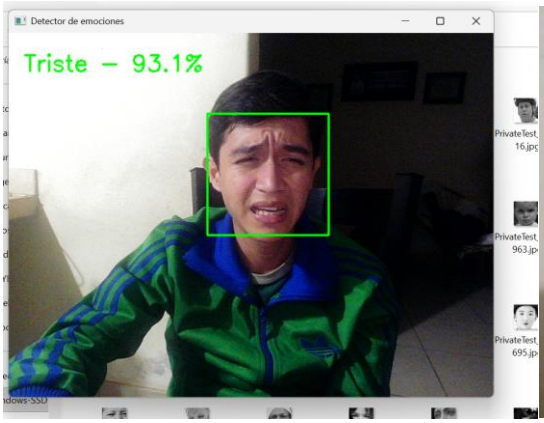
Muestra de resultados, también realizamos pruebas con un video:

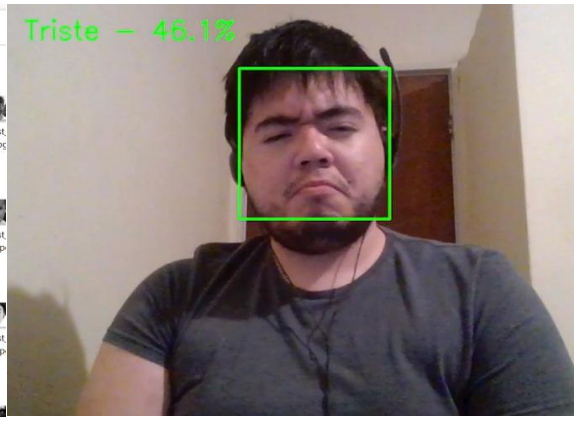
Felicidad, prueba 1 y 2



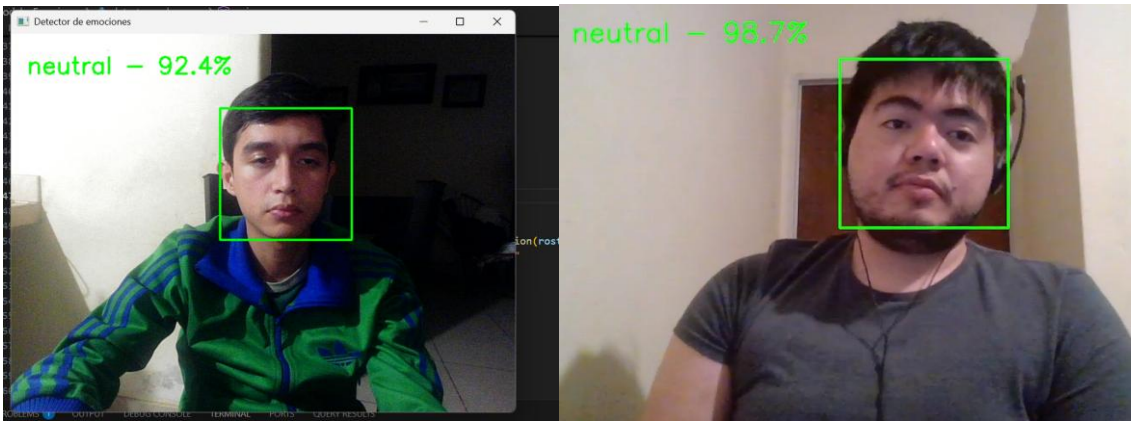


Tristeza prueba 1 y 2

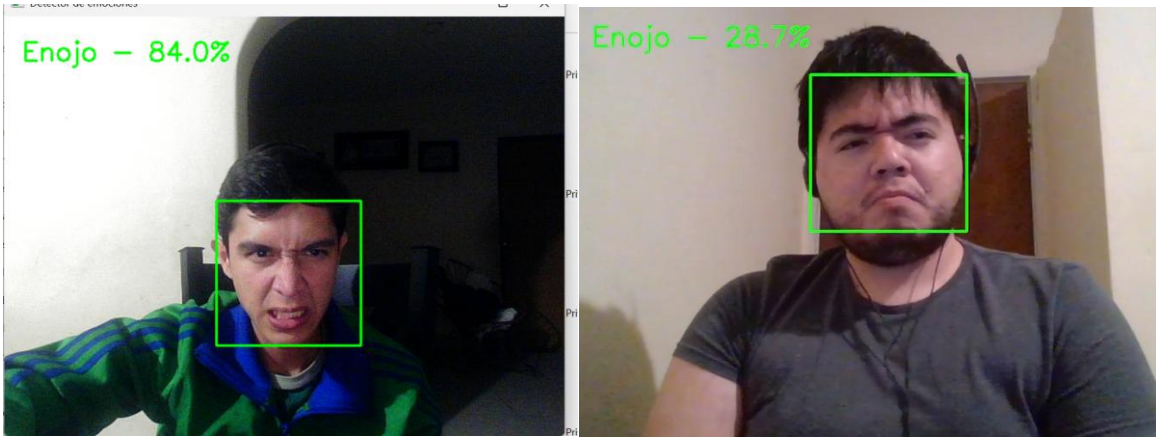




Neutral, prueba 1 y 2:



Enojo, prueba 1, 2 y 3:



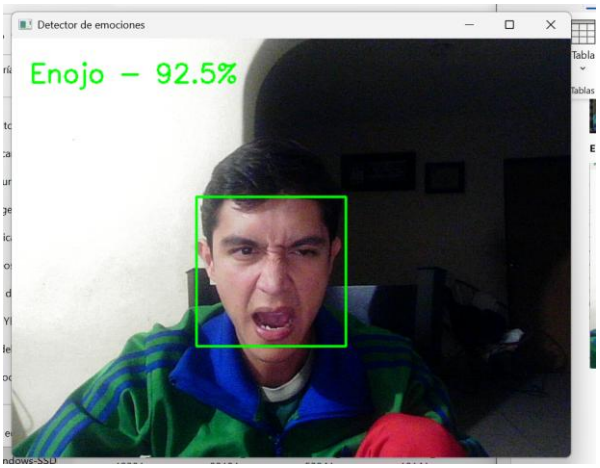


Tabla de resultados

Prueba	Emoción que se intento hacer	Emoción detectada	Confianza	Comentarios
1	Felicidad	Felicidad	99.8%	Sonrisa amplia
2	Felicidad	felicidad	97.7%	Sonrisa amplia
3	Tristeza	Triteza	93.1%	Gesticulación de tristeza amplia bastante expresiva
4	Tristeza	Tristeza	46.1%	gesticulación menos expresiva
5	Neutral	Neutral	92.4%	Gesticulación sin expresión
6	Neutral	Neutral	98.4%	Gesticulación sin expresión
7	Enojo	Enojo	84.0%	gesticulación con expresión un poco fuerte
8	Enojo	Enojo	28.87%	Gesticulación menos expresiva
10	Enojo	Enojo	92.5%	Gesticulacion bastante expresiva

Grafica

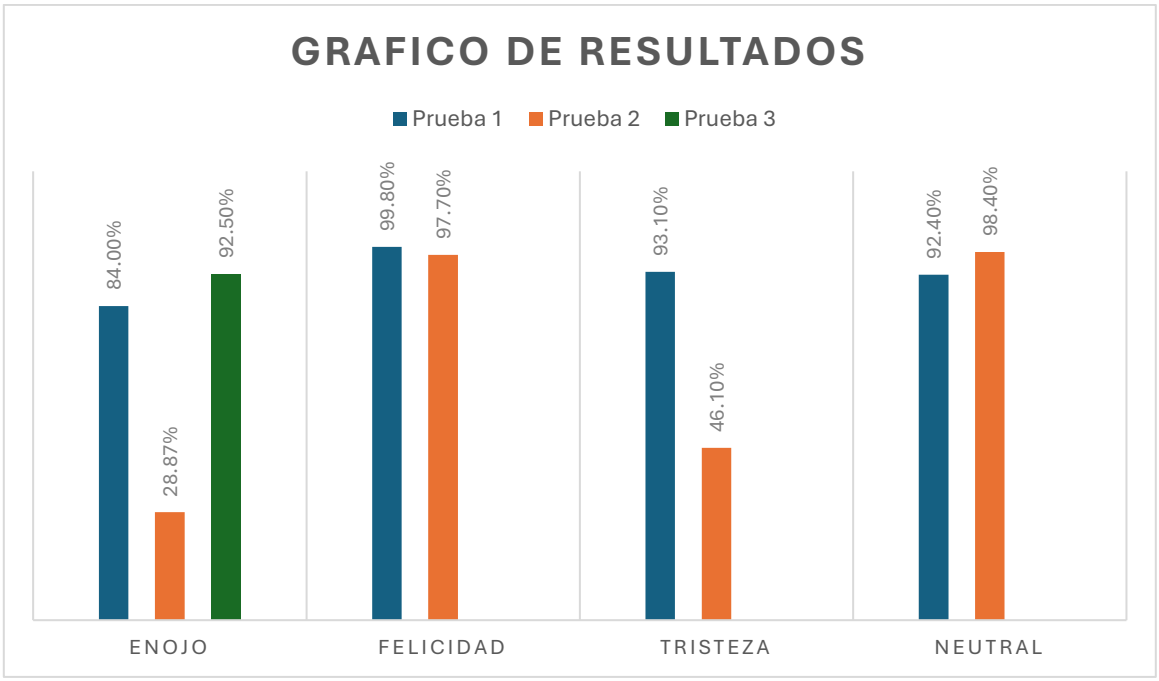


GRAFICO DE RESULTADOS
Prueba 1
Prueba 2
Prueba 3
84.00%
28.87%
92.50%
99.80%
97.70%
93.10%
46.10%
92.40%
98.40%
ENOJO
FELICIDAD
TRISTEZA
NEUTRAL

CONCLUSIONES

Fue un proyecto que fue bastante demandante, porque con forme nos pusimos manos a la obra con el proyecto, estuvimos evaluando diferentes opciones, entrenamiento propio de modelos de IA, diseño de arquitecturas, pero, realmente no es así de sencillo como parece, claro, si se quiere tener buenos resultados, fue una grata experiencia conocer mucho mas sobre el funcionamiento de los modelos de redes neuronales (CNN), Y detectores de rostros, es increíble como la tecnología esta tan avanzada, y eso que lo que hicimos no es nada comparado a lo que ya existe o lo que se vendrá.