



**TECNOLÓGICO
NACIONAL DE MÉXICO**



Instituto Tecnológico de Culiacán

Carrera: Ingeniería en Sistemas Computacionales

Materia: Inteligencia Artificial

Profesor: Zuriel Dathan Mora Félix

Tarea: Aplicación Reconocedor de Emociones

Grupo:

11:00 AM – 12:00 PM

Equipo:

García Pérez José Ángel

Verdugo Bermúdez Sebastián

Detector de Emociones con Deep Learning (FER2013)

Este proyecto implementa un sistema de detección de emociones faciales basado en técnicas de Deep Learning, específicamente utilizando redes neuronales convolucionales (CNN). El objetivo es identificar emociones en rostros humanos a partir de imágenes, ya sea estáticas o en tiempo real mediante una cámara web. El sistema fue entrenado con el conjunto de datos FER2013, ampliamente utilizado para este tipo de tareas en el ámbito de la visión por computadora.

Dataset Utilizado


El conjunto de datos utilizado para entrenar y evaluar el modelo es el FER2013. Este dataset contiene más de 35,000 imágenes faciales etiquetadas con una de las siete emociones básicas. Las imágenes son en escala de grises y tienen una resolución de 48x48 píxeles. El dataset está dividido en carpetas de entrenamiento y prueba.

Para la detección de rostros previa a la clasificación, se utiliza el archivo Haar Cascade `haarcascade_frontalface_default.xml`, una herramienta proporcionada por OpenCV que permite localizar rostros de manera rápida y eficiente.

FER-2013

Learn facial expressions from an image

[Data Card](#) [Code \(594\)](#) [Discussion \(8\)](#) [Suggestions \(6\)](#)



Fear Happy Neutral

About Dataset

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centred and occupies about the same amount of space in each image.

The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The training set consists of 28,709 examples and the public test set consists of 3,589 examples.

Usability ⓘ
7.50

License
[Database: Open Database, Cont...](#)

Expected update frequency
Not specified

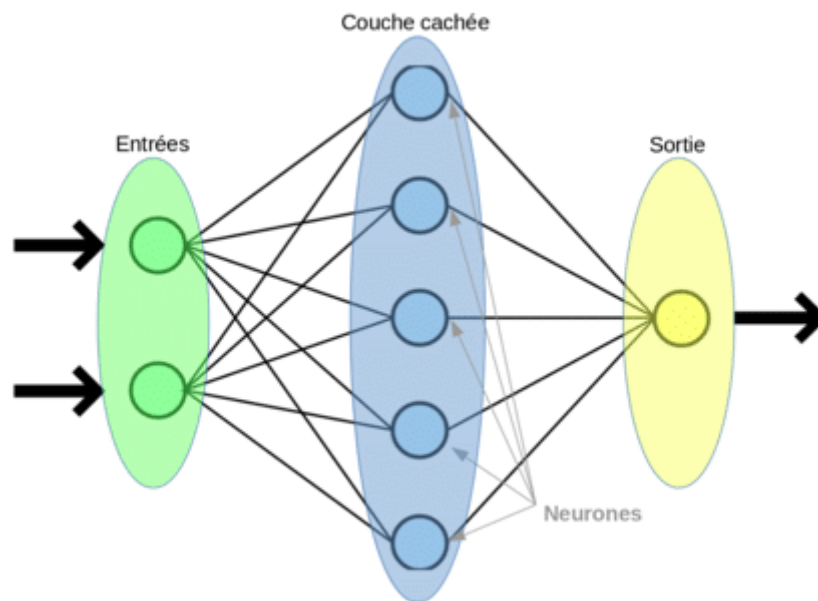
Tags

Seleccionar Arquitectura Neuronal

La red neuronal utilizada en este proyecto se denomina `Deep_Emotion`, y está diseñada específicamente para el reconocimiento de emociones a partir de imágenes en escala de grises. Esta red, implementada en PyTorch, recibe como entrada imágenes de 48x48 píxeles y devuelve como salida la predicción de una de las siete emociones básicas.

La arquitectura de la CNN incluye varias capas de convolución, activaciones ReLU, capas de pooling para reducción de dimensionalidad y capas completamente conectadas. También se utiliza dropout para prevenir el sobreajuste. El diseño está

enfocado en extraer de forma eficiente las características faciales relevantes para distinguir entre las siguientes emociones: angry, disgust, fear, happy, neutral, sad y surprise.



Definir Parámetros de Entrenamiento

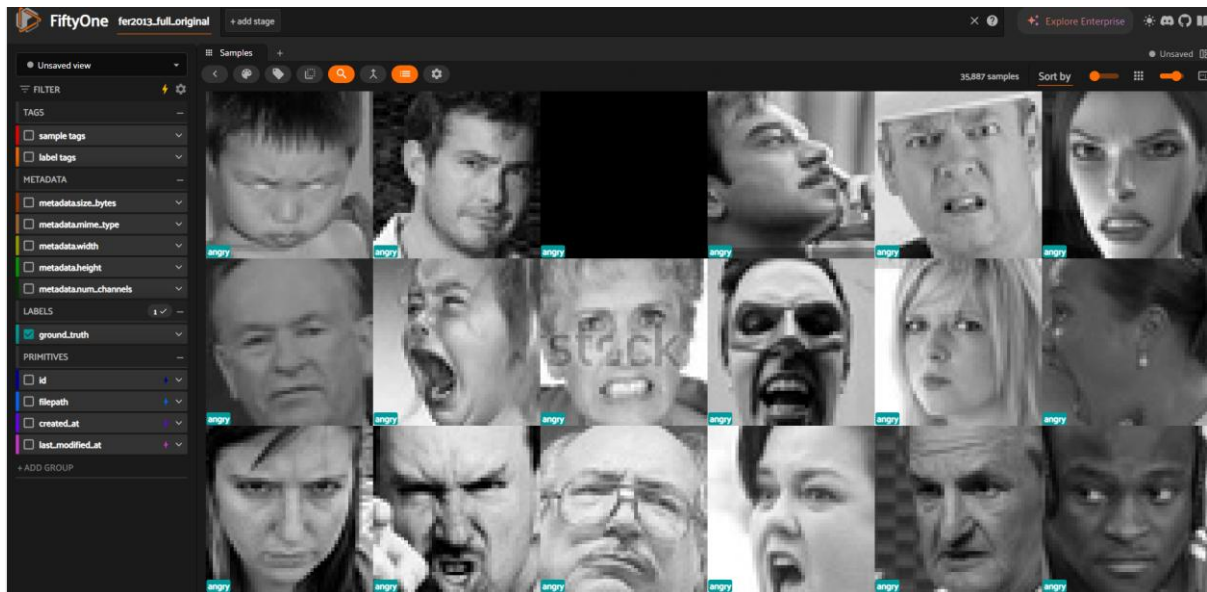
Los parámetros y configuraciones del entrenamiento están definidos en el archivo `config.py`. Este archivo contiene valores esenciales como la tasa de aprendizaje, el número de épocas, el tamaño del batch, y las rutas a los datasets. Esta centralización de la configuración facilita la modificación y reutilización del código en distintos entornos.

Antes del entrenamiento, el conjunto de datos es preprocesado utilizando el script `preProcesarDataset.py`. Este proceso incluye técnicas de aumento de datos aplicadas con la librería `Albumentations`, en combinación con `FiftyOne` para la organización y visualización del dataset. Entre las transformaciones aplicadas se encuentran:

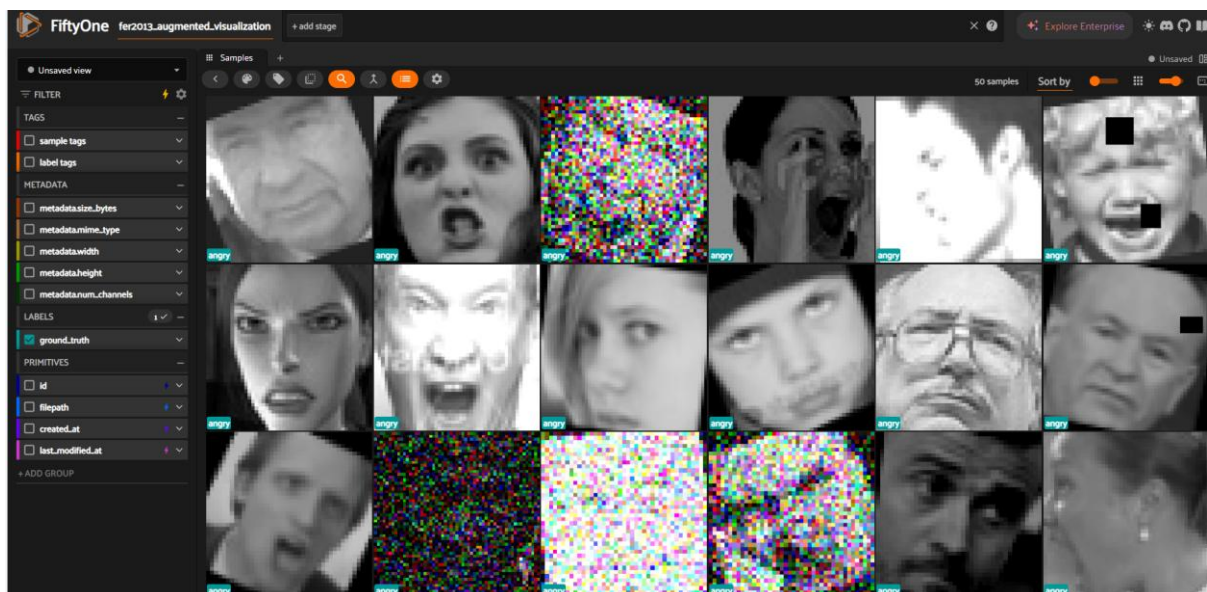
- Rotaciones y traslaciones.
- Variaciones de brillo y contraste.
- Cambios de escala y recortes aleatorios.

Estas técnicas buscan mejorar la capacidad del modelo para generalizar ante variaciones en las imágenes de entrada.

Aquí podemos visualizar las imágenes en fiftyone de nuestro dataset antes de las transformaciones.



En esta otra imagen podemos ver imágenes con las diversas transformaciones aplicadas durante el preprocesado.

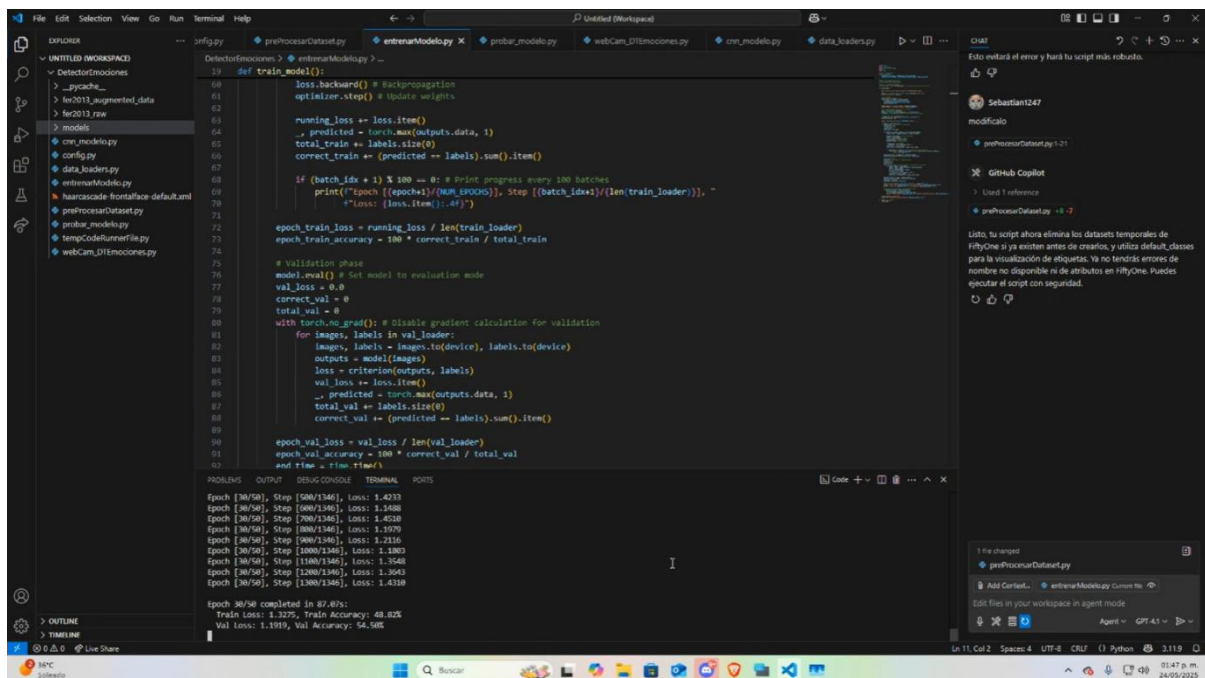


Entrenamiento del Modelo

El entrenamiento del modelo se realiza ejecutando el archivo `entrenarModelo.py`. En este script se inicializa el modelo `Deep_Emotion`, se cargan los datos preprocesados mediante los `DataLoaders` definidos en `data_loaders.py`, y se entrena el modelo utilizando una función de pérdida apropiada (`CrossEntropyLoss`) y un optimizador como Adam o SGD.

Durante el entrenamiento se evalúa el desempeño del modelo en cada época, monitoreando la pérdida y la precisión en el conjunto de validación. Una vez completado, los pesos del modelo son guardados automáticamente en la carpeta `models/`, lo que permite usarlos posteriormente sin necesidad de volver a entrenar desde cero.

Las siguientes son imágenes del proceso del entrenamiento del modelo:



The screenshot displays a code editor with the `entrenarModelo.py` file open. The script defines a `train_model()` function that handles the training loop, including data loading, model evaluation, and saving. The output window at the bottom shows the progress of the training over 30 epochs.

```
def train_model():
    loss.backward() # backpropagation
    optimizer.step() # update weights

    running_loss += loss.item()
    _, predicted = torch.max(outputs.data, 1)
    total_train += labels.size(0)
    correct_train += (predicted == labels).sum().item()

    if (batch_idx + 1) % 100 == 0: # Print progress every 100 batches
        print(f'epoch [{epoch+1}/{MAX_EPOCHS}], Step [{batch_idx+1}/{len(train_loader)}], "
              * Loss: {loss.item():.4f}"')

    epoch_train_loss = running_loss / len(train_loader)
    epoch_train_accuracy = 100 * correct_train / total_train

    # Validation phase
    model.eval() # Set model to evaluation mode
    val_loss = 0.0
    correct_val = 0
    total_val = 0

    with torch.no_grad(): # Disable gradient calculation for validation
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total_val += labels.size(0)
            correct_val += (predicted == labels).sum().item()

    epoch_val_loss = val_loss / len(val_loader)
    epoch_val_accuracy = 100 * correct_val / total_val
    and time = time.time()
```

Epoch [30/50], Step [1300/1340], Loss: 1.4213
Epoch [30/50], Step [900/1340], Loss: 1.3488
Epoch [30/50], Step [700/1340], Loss: 1.4010
Epoch [30/50], Step [500/1340], Loss: 1.3279
Epoch [30/50], Step [300/1340], Loss: 1.2116
Epoch [30/50], Step [1000/1340], Loss: 1.1803
Epoch [30/50], Step [1100/1340], Loss: 1.3044
Epoch [30/50], Step [1200/1340], Loss: 1.3641
Epoch [30/50], Step [1300/1340], Loss: 1.4319

epoch 30/50 completed in 87.87s:
Train Loss: 1.3275, Train Accuracy: 49.82%
Val Loss: 1.1919, Val Accuracy: 54.96%

```
def train_model():
    # Predictions
    predicted = torch.max(outputs.data, 1)
    total_val += labels.size(0)
    correct_val += (predicted == labels).sum().item()

    epoch_val_loss = val_loss / len(val_loader)
    epoch_val_accuracy = 100 * correct_val / total_val
    end_time = time.time()
    epoch_time = end_time - start_time

    print(f"Epoch {epoch+1}/{EPOCHS} completed in {epoch_time:.2f}s")
    print(f"Train loss: {epoch_train_loss:.4f}, Train Accuracy: {epoch_train_accuracy:.2f}%")
    print(f"Val loss: {epoch_val_loss:.4f}, Val Accuracy: {epoch_val_accuracy:.2f}%")

    # Save the best model based on validation accuracy
    if epoch_val_accuracy > best_val_accuracy:
        best_val_accuracy = epoch_val_accuracy
        # Save the best model
        torch.save(model.state_dict(), 'best_model.pth')

# Training progress
Epoch [40/50]: Step [900/1340], loss: 1.3828
Epoch [40/50]: Step [900/1340], loss: 1.4545
Epoch [40/50]: Step [1000/1340], loss: 1.3163
Epoch [40/50]: Step [1100/1340], loss: 1.3453
Epoch [40/50]: Step [1200/1340], loss: 1.4096
Epoch [40/50]: Step [1300/1340], loss: 1.3467

Epoch 40/50 completed in 87.85s:
Train loss: 1.3182, Train Accuracy: 49.44%
Val loss: 1.1754, Val Accuracy: 54.95%

Epoch [50/50]: Step [900/1340], loss: 1.3800
Epoch [50/50]: Step [1000/1340], loss: 1.6448
Epoch [50/50]: Step [1100/1340], loss: 1.5066
Epoch [50/50]: Step [1200/1340], loss: 1.4500
Epoch [50/50]: Step [1300/1340], loss: 1.2262
Epoch [50/50]: Step [900/1340], loss: 1.2752
Epoch [50/50]: Step [1000/1340], loss: 1.4203
Epoch [50/50]: Step [1100/1340], loss: 1.2124
Epoch [50/50]: Step [1200/1340], loss: 1.2101
Epoch [50/50]: Step [1300/1340], loss: 1.2318
Epoch [50/50]: Step [900/1340], loss: 1.2115
Epoch [50/50]: Step [1000/1340], loss: 1.3468
Epoch [50/50]: Step [1100/1340], loss: 1.3283

Epoch 50/50 completed in 91.22s:
Train loss: 1.3186, Train Accuracy: 49.30%
Val loss: 1.1781, Val Accuracy: 55.78%
Model saved to models/deep_emotion_best_model.pth with improved validation accuracy: 55.78%

Training complete!
Best validation accuracy achieved: 55.78%
PS D:\Sebastian\Documents\tec\cursos\InteligenciaArtificial\DetectorEmociones\
```

En estas imágenes se observa cómo el modelo aprende a lo largo de las "épocas", con mensajes que indican la "loss" (pérdida) y la "accuracy" (precisión) en cada iteración, reflejando su progreso y rendimiento a medida que procesa los datos de entrenamiento, al final se puede apreciar que se guardó el modelo con la mejor precisión de validación alcanzada (del **55.78%**), lo cual representa el mejor rendimiento logrado durante el entrenamiento.

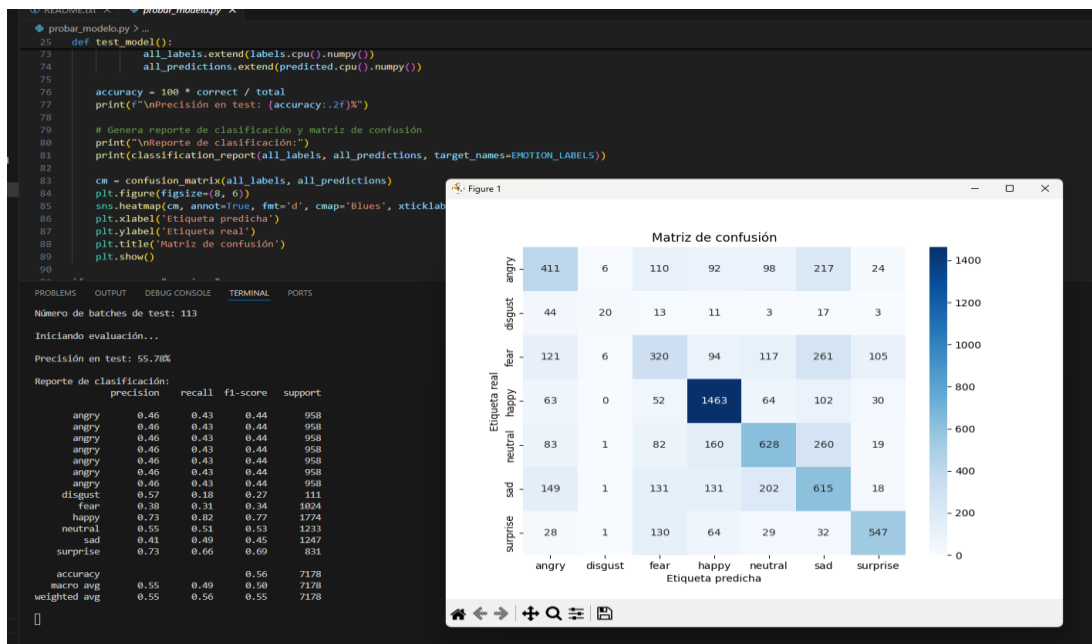
Evaluación con Matriz de Confusión

El script **probar_modelo.py** carga los pesos guardados del modelo entrenado y lo evalúa sobre el conjunto de prueba. Se calcula la precisión general, que en este caso es del **55.78%**, indicando el porcentaje de predicciones correctas.

Además, se genera un reporte de clasificación que muestra el rendimiento por clase de emoción (angry, disgust, fear, happy, neutral, sad, surprise) usando precisión, recall y F1-score. La precisión mide cuántas predicciones de una clase fueron correctas; por ejemplo, para "angry" es 0.46, es decir, el 46% de las veces que el modelo predijo "angry" fue correcto. El recall indica qué porcentaje de instancias reales de la clase fueron detectadas, y el F1-score combina ambas métricas para dar un balance.

La matriz de confusión, visualizada con Seaborn y Matplotlib, muestra en la diagonal los aciertos y fuera de ella los errores de clasificación. Por ejemplo, el modelo confunde frecuentemente "happy" con "disgust" o "sad", y "neutral" con "sad". Esto ayuda a identificar dónde el modelo tiene dificultades y orientar mejoras.

Nuestra matriz de confusión nos arrojó lo siguiente:



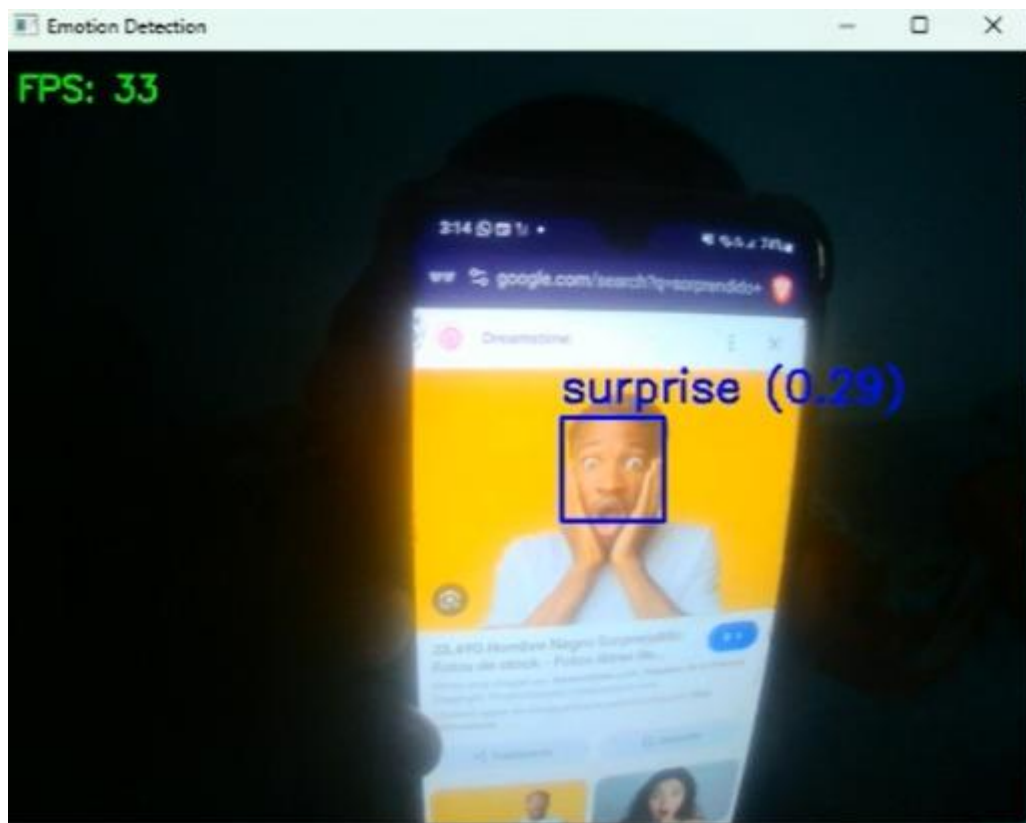
En esta imagen se puede apreciar la precisión del modelo (55.78% en este caso) y genera un reporte de clasificación que detalla métricas como precisión, recall y F1-score para cada emoción (enojo, disgusto, miedo, felicidad, neutral, tristeza, sorpresa). A la derecha, la Matriz de Confusión visualiza cómo el modelo acierta y se equivoca: los números en la diagonal principal (azules más oscuros) son predicciones correctas, mientras que los demás indican errores, mostrando qué emociones confunde el modelo entre sí.

Resultados del Reconocedor de Emociones

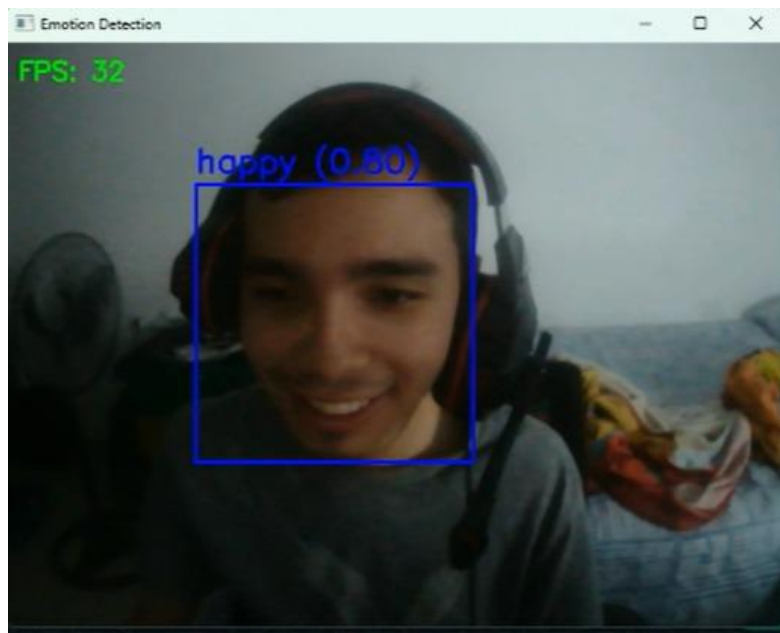
Una vez entrenado, el modelo puede ser utilizado para inferencia en tiempo real con el archivo `webCam_DTEmociones.py`. Este script utiliza una webcam para capturar video en vivo y emplea el clasificador Haar Cascade (`haarcascade_frontalface_default.xml`) de OpenCV para detectar rostros dentro del video.

Una vez detectado un rostro, se recorta, se redimensiona y se transforma para que pueda ser procesado por la red `Deep_Emotion`, la cual devuelve la emoción predicha. La emoción se muestra en tiempo real sobre el rostro detectado, permitiendo una visualización clara y dinámica del resultado.

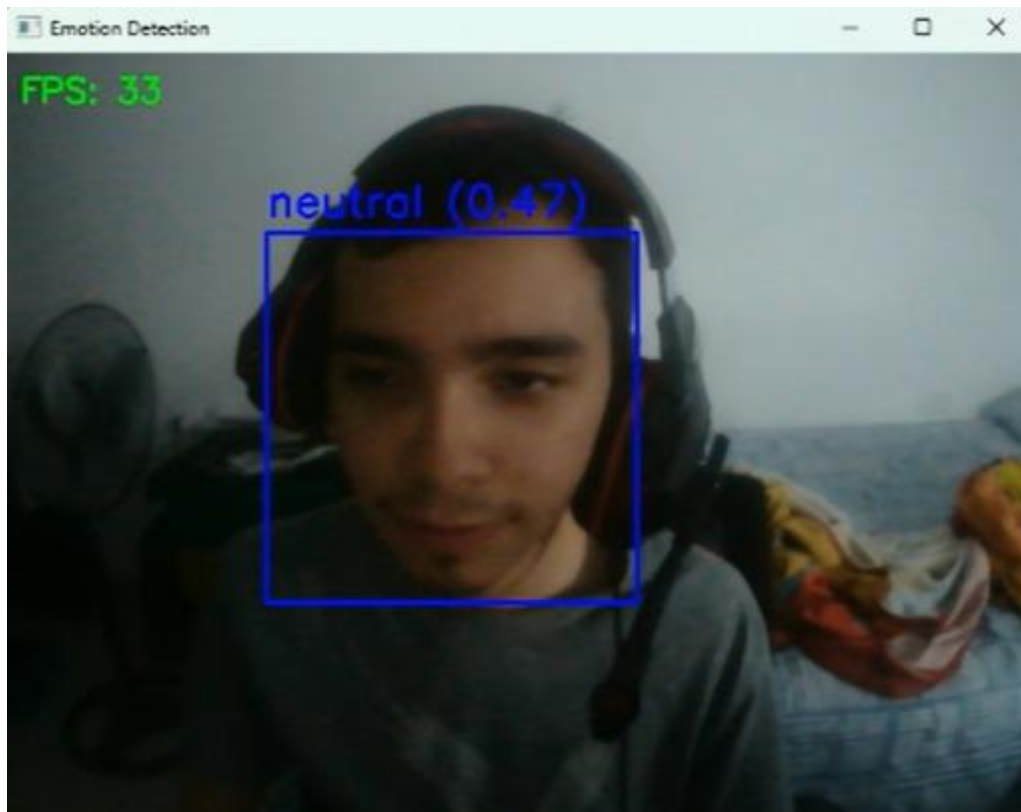
Probando el modelo obtuvimos lo siguiente:



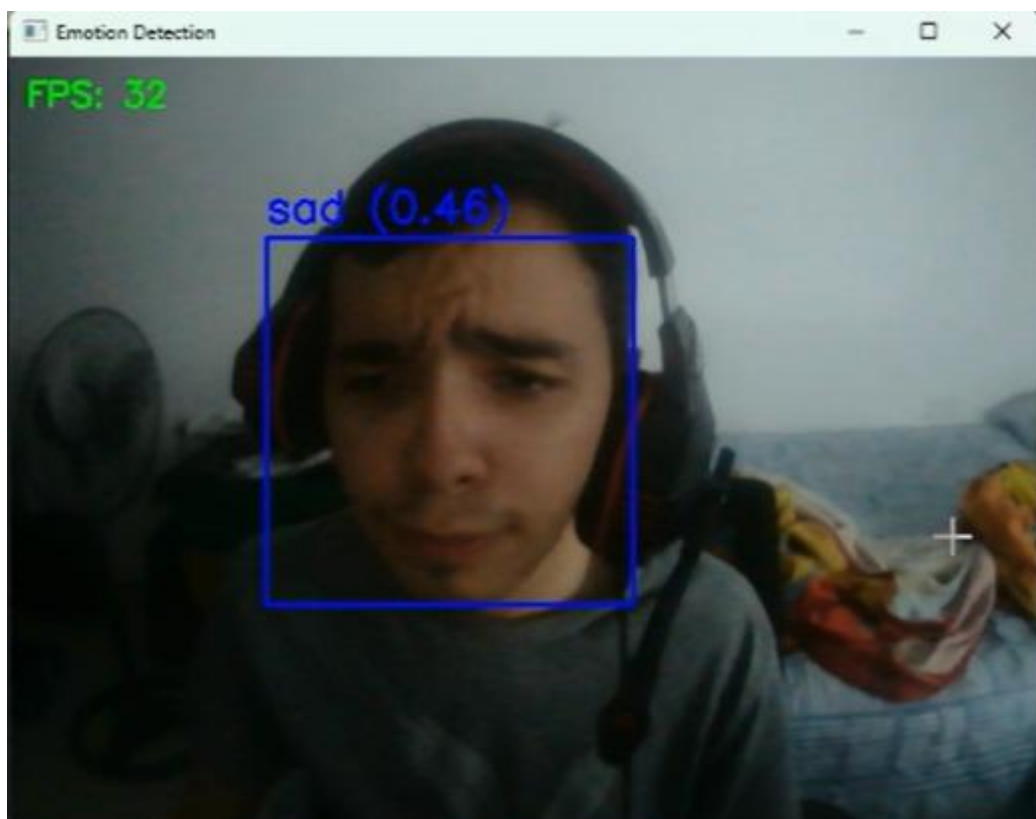
El sistema detecto un rostro, analizo sus características faciales y clasifico la emoción expresada como "surprise" (sorpresa) con alta confianza.



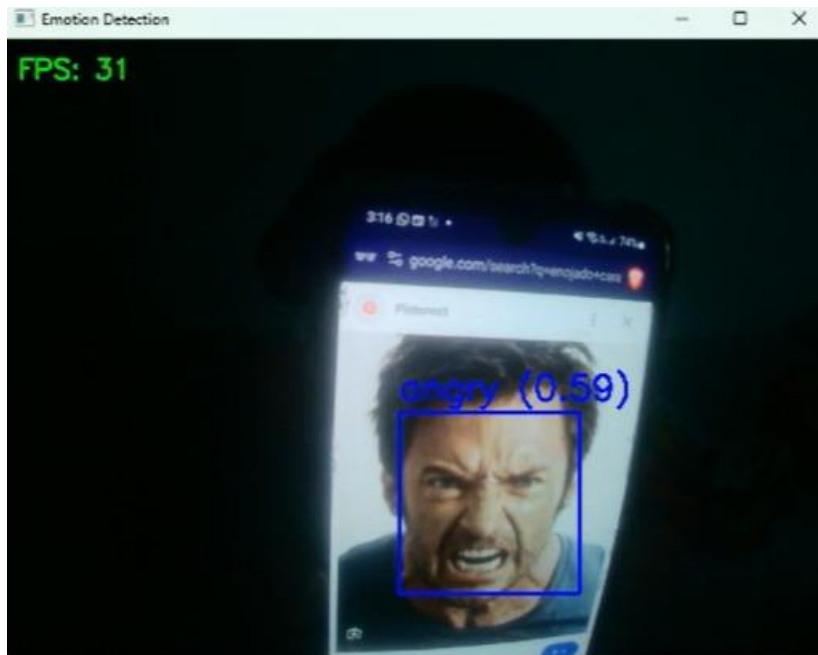
El sistema detecto un rostro, analizando sus características faciales y clasificando la emoción expresada como "happy" (felicidad) con un alto grado de confianza.



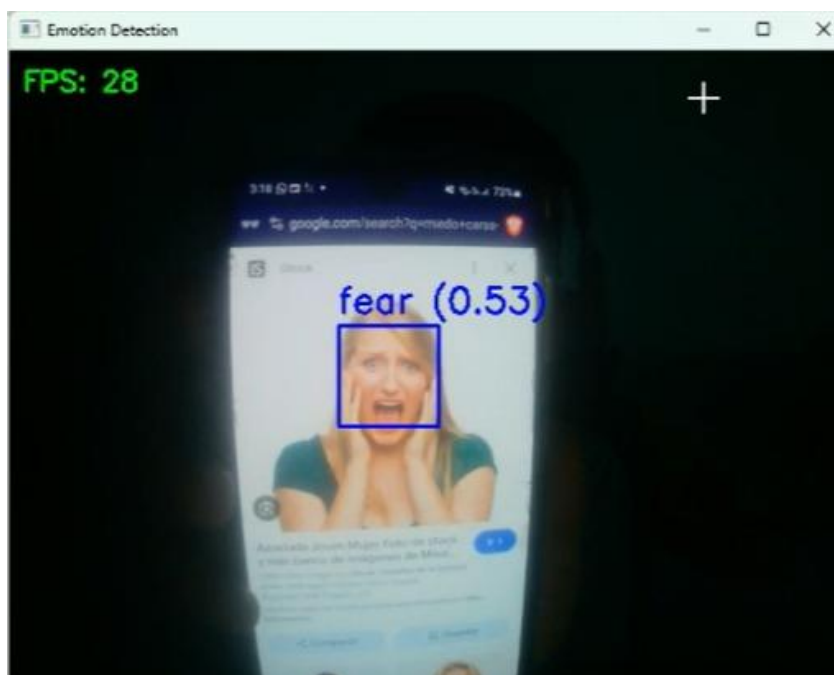
El sistema detecto un rostro, analizo sus características faciales y clasifico la emoción expresada como "neutral" con una confianza moderada.



El sistema detecto un rostro, analizo sus características faciales y clasifico la emoción expresada como "sad" (triste) con una confianza moderada.



El sistema detecta un rostro, analizo sus características faciales y clasifico la emoción expresada como "angry" (enojado) con una confianza del 59%.



El sistema detecta un rostro en la pantalla de un teléfono, analiza sus características faciales y clasifica la emoción expresada como "fear" (miedo) con una confianza del 53%.

Link del video de prueba de cámara:

<https://youtu.be/pevQbXvKPuc>

Conclusión

Este proyecto fue una buena oportunidad para aplicar los conocimientos de redes neuronales y visión por computadora en un caso práctico: la detección de emociones faciales. Utilizando una red convolucional personalizada y el dataset FER2013, se logró entrenar un modelo capaz de reconocer siete emociones básicas a partir de imágenes en escala de grises.

Durante la validación del modelo se obtuvo una precisión del 56%, lo cual, si bien no es un resultado perfecto, es un buen punto de partida considerando la complejidad del problema y las limitaciones del dataset. Además, se integró la detección de emociones en tiempo real con cámara web, lo que demuestra la funcionalidad del modelo fuera de un entorno controlado.

Este trabajo también permitió entender mejor todo el flujo de un proyecto de inteligencia artificial, desde el preprocesamiento de datos, el entrenamiento y la evaluación del modelo, hasta su implementación práctica. Aunque hay margen para mejorar, por ejemplo, ajustando los hiperparámetros o ampliando el dataset, el sistema desarrollado cumple con su objetivo y sirve como base para futuros avances.