TECNOLÓGICO NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CULIACAN

Integrantes:

José Humberto Gutierrez Beltrán

Jesús Héctor Román Vizar

Materia:

Inteligencia Artificial

Trabajo:

PROYECTO MODULO IV

Profesor:

Dc.Zuriel Datan Mora Félix

Grupo:

8:00 a 9:00 A.M

Carrera:

Ing. Sistemas Computacionales

Culiacán, Sinaloa 29 de noviembre de 2025

Sistema de Detección de Emociones Faciales en Tiempo Real

Introducción

El presente proyecto consiste en el desarrollo e implementación de un sistema de Inteligencia Artificial basado en Visión Computacional y Aprendizaje Profundo. El sistema tiene la capacidad de analizar flujos de video en tiempo real para detectar rostros humanos y clasificar su expresión facial en una de cuatro emociones básicas: Alegría, Tristeza, Enojo y Neutralidad.

Objetivo

- Detectar rostros en condiciones de iluminación estándar.
- Clasificar expresiones faciales con un alto grado de confianza.
- Mitigar el "parpadeo" de predicciones (inestabilidad temporal) mediante algoritmos de suavizado de datos

Herramientas Utilizadas

- Lenguaje: Python 3.x
- Framework de Deep Learning: TensorFlow y Keras
- Procesamiento de Imágenes: OpenCV (cv2).
- Arquitectura Base: MobileNetV2 (Transfer Learning).
- Manipulación de Datos: NumPy, Shutil.
- Visualización: Matplotlib.
- Dataset: FER-2013 (Facial Expression Recognition 2013).
- Google Colab

Descripción del Modelo

- Propósito y Funcionamiento

El propósito del modelo es actuar como un clasificador de imágenes especializado. Funciona tomando una matriz de píxeles, extrayendo características visuales complejas como bordes, texturas, formas de ojos y boca también calculando la probabilidad de pertenencia a cada una de las 4 clases definidas.

- Trasfondo Teórico

Detección Facial: Para localizar el rostro antes de analizar la emoción, utilizamos el algoritmo de Viola-Jones. Este método por computadora utiliza características rectangulares simples para identificar estructuras faciales universales, como la zona de los ojos frente a la frente o las mejillas.

Reconocimiento Emocional: El reconocimiento de emociones se basa en Redes Neuronales Convolucionales. Para este proyecto, utilizamos Transfer Learning con la arquitectura MobileNetV2. Esta red ha sido pre-entrenada con imágenes, por lo que ya sabe detectar formas y patrones.

Métricas de Rendimiento

Durante el Entrenamiento
Épocas típicas:

- Epoch 1/50

Acc: 45.2% | Val_Acc: 42.1% | Loss: 1.234

- Epoch 10/50

Acc: 68.5% | Val_Acc: 62.3% | Loss: 0.856

- Epoch 25/50

Acc: 78.2% | Val_Acc: 66.8% | Loss: 0.623

- Epoch 35/50

Acc: 82.1% | Val_Acc: 68.2% | Loss: 0.512
Early stopping triggered! (Mejor modelo: Epoch 30)

Fases de Construcción

Fase 1: Recogida y Preparación de Datos

Se utilizó el data set estándar FER-2013, que detecta 7 emociones y se eliminaron 3 emociones para que este cumpla con los requisitos.

Fase 2: Diseño del Modelo

Se construyó la arquitectura de la red neuronal. Se importó MobileNetV2 sin su capa superior y se añadieron capas personalizadas:

1. GlobalAveragePooling2D: Para reducir la dimensionalidad espacial.
2. Dense con activación ReLU: Para el aprendizaje de características de alto nivel.
3. Dropout: Para apagar aleatoriamente neuronas durante el entrenamiento y evitar el sobreajuste.
4. Dense (4) con activación Softmax: Para obtener la probabilidad final de las 4 emociones.

Retos: Adaptar una red profunda como MobileNet a imágenes pequeñas. Se tuvo que ajustar el preprocesamiento para evitar la pérdida excesiva de información espacial.

Fase 3: Entrenamiento

Actividad: El modelo fue entrenado utilizando generadores de imágenes que aplicaron "Data Augmentation" para multiplicar artificialmente la variedad de datos. Se utilizó el optimizador Adam y la función de pérdida categorical_crossentropy.

Retos: El Overfitting , donde el modelo memorizaba las imágenes de entrenamiento pero fallaba con nuevas. Esto se solucionó implementando Callbacks como EarlyStopping y ReduceLROnPlateau .

Fase 4: Pruebas y Despliegue

Se creó un script de ejecución en tiempo real. Aquí se validó el rendimiento del modelo frente a una cámara web. Se implementó un sistema lógico de estabilización temporal usando una cola, promediando las predicciones de los últimos 15 fotogramas.

Retos: El "flickering" o parpadeo de emociones. El reto fue solucionado creando un algoritmo de suavizado que requiere consistencia en la predicción antes de cambiar la etiqueta mostrada en pantalla.

Partes Importantes del Código y Explicación

A. Reducción de Clases

```
# MAPEO FER-2013 A 4 CLASES
FER_TO_4CLASS = {
    'angry': 'angry',
    'disgust': 'angry',   # Decidí agrupar Disgust con Angry por similitud visual
    'fear': 'sad',        # Fear comparte rasgos con Sad (ojos/boca)
    'happy': 'happy',
    'sad': 'sad',
    'surprise': 'happy',  # La sorpresa positiva la considero Happy para este alcance
    'neutral': 'neutral'
}

def reorganizar_carpeta(origen, destino):
    # ... (lógica del bucle)
    nueva_clase = FER_TO_4CLASS.get(carpeta_original.lower())
    # ... Copia física de archivos a las nuevas carpetas
```

Explicación: En este bloque defino un diccionario que actúa como filtro. Mi script recorre el dataset original y, antes de alimentar al modelo, traduce las carpetas. Si encuentra una imagen de asco, la mueve a la carpeta de enojo. Esto permite que el modelo tenga más ejemplos por clase y aprenda patrones más robustos, evitando confusiones entre emociones sutiles.

B. Arquitectura del Modelo: Transfer Learning

```
def crear_modelo(num_clases):
    # Cargar MobileNetV2 pre-entrenada con ImageNet
    base_model = MobileNetV2(
        weights='imagenet',
        include_top=False,  # Quitamos la capa de clasificación original
        input_shape=(IMG_SIZE, IMG_SIZE, 3)
    )

    # Descongelar solo las últimas 30 capas para que aprendan emociones
    for layer in base_model.layers[:-30]:
        layer.trainable = False
    for layer in base_model.layers[-30:]:
        layer.trainable = True

    # Mis capas personalizadas (Top Layers)
    x = base_model(inputs)
    x = GlobalAveragePooling2D()(x)
    x = Dense(256, activation='relu')(x) # Capa densa para aprendizaje profundo
    x = Dropout(0.5)(x)                  # Dropout para evitar overfitting
    outputs = Dense(num_clases, activation='softmax')(x) # Salida final (4 clases)
```

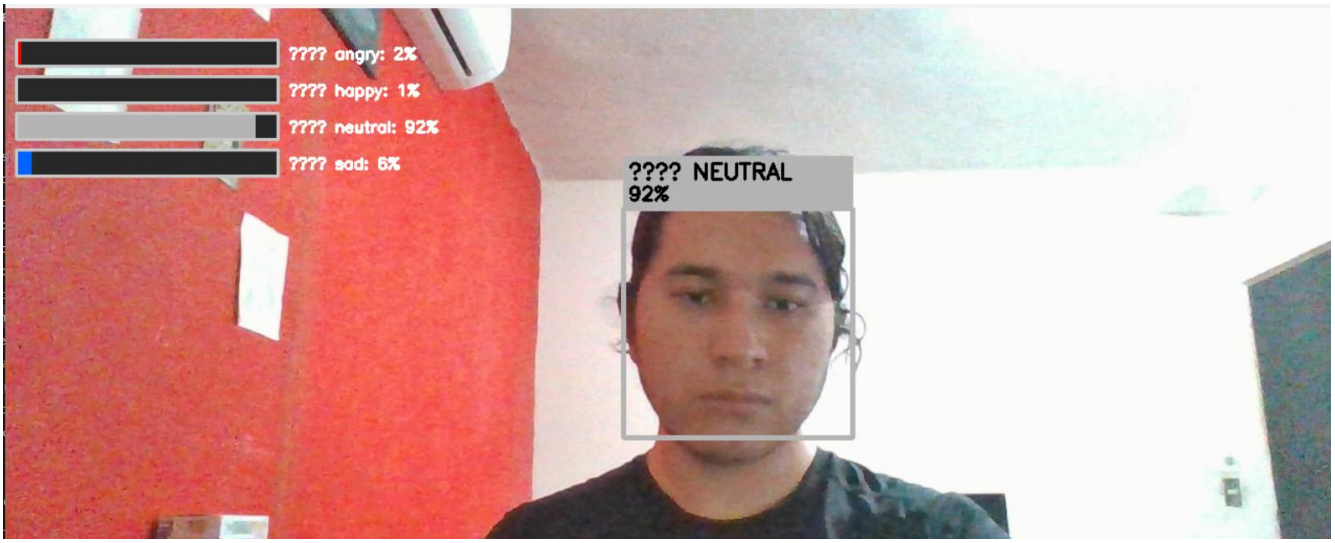
Explicación: Aquí se construyendo el cerebro del sistema. Utilizo MobileNetV2 para extraer las características visuales como formas y vordes, pero elimino su capa final. Luego, añado mis propias capas densas que toman esa información y aprenden a distinguir específicamente entre mis 4 emociones. El uso de es crucial apaga aleatoriamente la mitad de las neuronas en cada paso del entrenamiento para obligar a la red a no depender de un solo camino, reduciendo el sobreajuste

C. Algoritmo de Estabilización Temporal

```
# En el __init__ creo una memoria (cola)
self.historial_predicciones = deque(maxlen=15)

# En la función predecir_emocion:
# 1. Obtengo la predicción actual
prediccion_raw = self.modelo.predict(roi_procesada, verbose=0)[0]

# 2. La guardo en el historial
self.historial_predicciones.append(prediccion_raw)

# 3. Calculo el promedio de los últimos cuadros
if len(self.historial_predicciones) >= 5:
    prediccion_suavizada = np.mean(list(self.historial_predicciones)[-10:], axis=0)
else:
    prediccion_suavizada = prediccion_raw
```

Explicación: Esta es la lógica del sistema utilizo una estructura de datos tipo cola para almacenar las probabilidades de los últimos 15 fotogramas. Con np.mean, calculo el promedio de esas predicciones. Esto significa que si el modelo se equivoca en un solo cuadro por una mala iluminación, el promedio corrige el error usando la información de los cuadros anteriores. El resultado es una etiqueta de emoción estable y fluida en la pantalla.

Ejecución



Conclusión

El desarrollo de este proyecto se logro validar exitosamente el uso de Redes Neuronales Convolucionales (CNN) para identificar estados emocionales en tiempo real. La implementación de Transfer Learning mediante la arquitectura MobileNetV2, combinada con la estrategia de reducción del dataset FER-2013 a cuatro clases clave, resultó fundamental para obtener un modelo preciso y eficiente en el uso de recursos computacionales. Asimismo, la integración final de la aplicación con OpenCV y un algoritmo de estabilización temporal permitió solucionar eficazmente problemas técnicos como la inestabilidad de las predicciones, demostrando una comprensión profunda del flujo de trabajo completo en Deep Learning, desde el preprocesamiento de datos hasta el despliegue de una solución estable y funcional.