TECNOLÓGICO NACIONAL DE MÉXICO, TECNOLOGICO DE CULIACAN

Carrera:
Ingeniería en Sistemas Computacionales

Inteligencia Artificial
11:00 – 12:00

Sistema Clasificador de Emociones

Nombres:
Osuna Russell Ana Isabel
Rodríguez Valerio Jesús Ricardo

Docente:
Zuriel Dathan Mora Félix

29 de Mayo de 2025

1. Introducción y Objetivo del Proyecto

El objetivo de este proyecto es desarrollar un sistema capaz de clasificar expresiones faciales humanas en siete categorías de emociones (sorprendido, miedo, disgusto, feliz, triste, enojado, serio) utilizando una cámara web en tiempo real. Para ello, entrenamos una red neuronal con un dataset de imágenes de rostros etiquetados con estas emociones.

2. Herramientas y Librerías

- **Lenguaje de Programación:** Python
- **Librería de Deep Learning:** PyTorch
- **Procesamiento de Imágenes y Video:** OpenCV-Python
- **Manipulación Numérica:** NumPy
- **Carga de Datos y Transformaciones:** torchvision

3. Dataset

- **Nombre:** DATASET
- **Descripción:** El dataset consiste en imágenes de rostros que expresan siete categorías de emociones. (Ya presentado en la pasada entrega)

Estructura de Carpetas: Las imágenes están organizadas en una carpeta principal llamada DATASET. Dentro de esta, se espera una subcarpeta train y una subcarpeta test, cada una conteniendo 7 subcarpetas numeradas del '1' al '7', donde cada número corresponde a una emoción específica.

- **Mapeo de Clases (Emociones):**
    - '1': 'sorprendido'
    - '2': 'miedo'
    - '3': 'disgustado'
    - '4': 'feliz'
    - '5': 'triste'
    - '6': 'enojado'
    - '7': 'serio'
- **Número de Clases:** 7

4. Arquitectura Neuronal Recomendable

Para la tarea de clasificación de imágenes de expresiones faciales, elegimos **Red Neuronal Convolucional (CNN)**. Específicamente, **ResNet18 pre-entrenada en el dataset ImageNet** dado que es ideal para el modelo que realizamos.

- **Justificación de ResNet18:**
    - **Rendimiento Comprobado:** Las arquitecturas ResNet (Residual Networks) demuestran un excelente rendimiento en tareas de visión por computadora, como la clasificación de imágenes.
    - **Manejo de Profundidad:** Las conexiones residuales permiten entrenar redes más profundas de manera efectiva, mitigando el problema del gradiente desvaneciente o la dificultad encontrada para entrenar redes neuronales

artificiales mediante métodos de aprendizaje basados en descenso estocástico de gradientes y de retropropagación.
    - **Transfer Learning:** Al utilizar un modelo pre-entrenado en ImageNet, el modelo ya ha aprendido características visuales de bajo y medio nivel (bordes, texturas, formas básicas). Este conocimiento se puede transferir a nuestra tarea específica de clasificación de emociones, lo que permite un entrenamiento más rápido y con menos datos, y a menudo conduce a una mejor generalización. ResNet18 ofrece un buen equilibrio entre profundidad (y por tanto capacidad de aprendizaje) y eficiencia computacional en comparación con modelos más grandes como ResNet50 o ResNet101.

- **Modificación para Transfer Learning:**
  La arquitectura ResNet18 original, pre-entrenada en ImageNet, tiene una capa final completamente conectada diseñada para clasificar 1000 clases. Para adaptarla a nuestro problema de 7 emociones, esta última capa se reemplaza por una nueva capa completamente conectada con 7 neuronas de salida (una por cada emoción) y una función de activación Softmax implícita (manejada por la función de pérdida CrossEntropyLoss durante el entrenamiento). El resto de las capas convolucionales del modelo pre-entrenado se "congelan" inicialmente (pesos no actualizados) para preservar el conocimiento aprendido, y solo se entrenan los pesos de la nueva capa clasificadora.

5. Parámetros e Hiperparámetros de Entrenamiento

- **Parámetros del Modelo (aprendidos durante el entrenamiento):**
    - Son los pesos (weights) y sesgos (biases) de las neuronas en la red, especialmente los de la nueva capa clasificadora añadida y, si se realiza fine-tuning, los de las capas convolucionales superiores descongeladas.
- **Hiperparámetros (configurados antes del entrenamiento):**
    - **Tasa de Aprendizaje (Learning Rate):** 0.001. Determina el tamaño del paso que da el optimizador para ajustar los pesos del modelo. Un valor común para empezar con el optimizador Adam.
    - **Optimizador:** Adam. Es un algoritmo de optimización eficiente que adapta la tasa de aprendizaje para cada parámetro individualmente. Es una elección robusta y popular.
    - **Función de Pérdida (Loss Function):** CrossEntropyLoss. Es la función de pérdida estándar para problemas de clasificación multiclase. Combina una capa LogSoftmax y una NLLLoss (Negative Log Likelihood Loss) en una sola clase, lo que la hace numéricamente más estable.
    - **Número de Épocas (Epochs):** 15 (inicialmente, se puede ajustar). Una época representa una pasada completa del algoritmo de entrenamiento sobre todo el conjunto de datos de entrenamiento.
    - **Tamaño del Lote (Batch Size):** 16. Es el número de muestras de entrenamiento que se propagan a través de la red antes de actualizar los

pesos. Un tamaño de lote más grande puede llevar a una convergencia más rápida pero requiere más memoria.

- **División de Datos:** Se utiliza una división implícita mediante las carpetas train/ y val/ proporcionadas en el dataset. Típicamente, se busca una proporción de 70-80% para entrenamiento y 20-30% para validación.
- **Aumento de Datos (Data Augmentation) para el conjunto de entrenamiento:**
  - RandomResizedCrop(224): Recorta una región aleatoria de la imagen y la redimensiona a 224x224. Ayuda al modelo a ser robusto a variaciones en la escala y posición del objeto.
  - RandomHorizontalFlip(): Voltea la imagen horizontalmente con una probabilidad del 50%. Ayuda a generalizar para rostros orientados de diferentes maneras.
  - Grayscale(num_output_channels=3): Convierte la imagen a escala de grises y luego la replica en 3 canales para que sea compatible con la entrada de ResNet18 (que espera 3 canales). Esto es útil si las imágenes de entrada son en escala de grises, ya que la información de color puede no ser crucial para la clasificación de emociones y simplifica el modelo.
- **Transformaciones para el conjunto de validación/prueba:**
  - Resize(256): Redimensiona la imagen a 256x256.
  - CenterCrop(224): Recorta la región central de 224x224.
  - Grayscale(num_output_channels=3): Misma transformación que en entrenamiento para consistencia.
- **Normalización:** Todas las imágenes (entrenamiento, validación y prueba) se normalizan utilizando las medias y desviaciones estándar del dataset ImageNet: Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]). Esto ayuda a que los valores de entrada estén en un rango similar, lo que puede mejorar la convergencia del entrenamiento.

6. Implementación y Código Fuente (PyTorch)

```
import torch

import torch.nn as nn

import torch.optim as optim

from torchvision import datasets, models, transforms

import os

import cv2

import numpy as np

import time
```

```
#Configuracion de parametros
DATA_DIR = 'DATASET'
MODEL_SAVE_PATH = 'saved_models/emotion_classifier_resnet18.pth'
NUM_CLASSES = 7
BATCH_SIZE = 16
NUM_EPOCHS = 15
LEARNING_RATE = 0.001

SENTIMENT_MAP = {
   '2': 'miedo',
   '3': 'disgustado',
   '1': 'sorprendido',
   '2': 'miedo',
   '3': 'disgustado',
   '4': 'feliz',
   '5': 'triste',
   '6': 'enojado',
   '7': 'serio'
}

# Crear directorio para guardar modelos
os.makedirs('saved_models', exist_ok=True)

# En caso de usar una grafica, configurar el dispositivo
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(f"Usando dispositivo: {device}")


# --- 1. PREPARACIÓN DE DATOS ---
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.Grayscale(num_output_channels=3),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
0.225])
    ]),
    'test': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.Grayscale(num_output_channels=3),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
0.225])
    ]),
}


try:
    print(f"Cargando datos desde: {DATA_DIR}")
```

```
    image_datasets = {x: datasets.ImageFolder(os.path.join(DATA_DIR, x),
data_transforms[x])

                      for x in ['train', 'test']}

    # Separar los datos para entrenamiento y prueba

    num_workers_setting = 4 if os.name != 'nt' else 0

    print(f"Usando num_workers = {num_workers_setting} para
DataLoaders.")



    dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x],
batch_size=BATCH_SIZE,

                                                  shuffle=True,
num_workers=num_workers_setting)

                  for x in ['train', 'test']}

    dataset_sizes = {x: len(image_datasets[x]) for x in ['train',
'test']}



    class_names = image_datasets['train'].classes

    print(f"Clases detectadas (nombres de carpeta): {class_names}")

    print(f"Tamaños de dataset: Train={dataset_sizes['train']},
test={dataset_sizes['test']}")



except FileNotFoundError:

    print(f"Error: Directorio de datos no encontrado.")

    print(f"Asegúrate de que la carpeta '{DATA_DIR}' exista y contenga
las subcarpetas 'train' y 'test'.")

    print(f"Dentro de 'train' y 'test', deben estar las carpetas '1' a
'{NUM_CLASSES}'.")

    exit()
```

```
except Exception as e:
    print(f"Ocurrió un error al cargar los datos: {e}")
    exit()



# --- 2. DEFINICIÓN DEL MODELO (ResNet18 pre-entrenado) ---
def get_model(num_classes_out):
    # Cargar ResNet18 pre-entrenado en ImageNet
    model =
models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1)

    for param in model.parameters():
        param.requires_grad = False

    num_ftrs = model.fc.in_features
    model.fc = nn.Linear(num_ftrs, num_classes_out)

    model = model.to(device)
    return model

# --- 3. FUNCIÓN DE ENTRENAMIENTO ---
def train_model(model, criterion, optimizer, num_epochs=25):
    since = time.time()
    best_model_wts = model.state_dict()
    best_acc = 0.0
```

```python
for epoch in range(num_epochs):
    print(f'Epoch {epoch+1}/{num_epochs}')
    print('-' * 10)

    for phase in ['train', 'test']:
        if phase == 'train':
            model.train()
        else:
            model.eval()

        running_loss = 0.0
        running_corrects = 0

        for inputs, labels in dataloaders[phase]:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)
```

```
                    if phase == 'train':

                        loss.backward()

                        optimizer.step()



                running_loss += loss.item() * inputs.size(0)

                running_corrects += torch.sum(preds == labels.data)



            epoch_loss = running_loss / dataset_sizes[phase]

            epoch_acc = running_corrects.double() / dataset_sizes[phase]



            print(f'{phase} Loss: {epoch_loss:.4f} Acc:
{epoch_acc:.4f}')



            if phase == 'test' and epoch_acc > best_acc:

                best_acc = epoch_acc

                best_model_wts = model.state_dict()

                torch.save(model.state_dict(), MODEL_SAVE_PATH)

                print(f"Nuevo mejor modelo guardado en {MODEL_SAVE_PATH}
con test Acc: {best_acc:.4f}")

        print()



    time_elapsed = time.time() - since

    print(f'Entrenamiento completo en {time_elapsed // 60:.0f}m
{time_elapsed % 60:.0f}s')

    print(f'Mejor Val Acc: {best_acc:4f}')
```

```
    model.load_state_dict(best_model_wts)

    return model



# --- 4. INICIAR ENTRENAMIENTO ---

def start_training():

    print(f"Iniciando entrenamiento para {NUM_CLASSES} emociones...")

    model_ft = get_model(NUM_CLASSES)



    criterion = nn.CrossEntropyLoss()



    optimizer_ft = optim.Adam(filter(lambda p: p.requires_grad,
model_ft.parameters()), lr=LEARNING_RATE)



    trained_model = train_model(model_ft, criterion, optimizer_ft,
num_epochs=NUM_EPOCHS)

    print(f"Modelo entrenado y el mejor modelo guardado en
{MODEL_SAVE_PATH}")



# --- 5. PRUEBAS CON WEBCAM ---

def test_with_webcam():

    print("Iniciando prueba con webcam...")

    if not os.path.exists(MODEL_SAVE_PATH):

        print(f"Modelo no encontrado en {MODEL_SAVE_PATH}. Por favor,
entrena el modelo primero.")

        return

    if not class_names:
```

```
        print("Error: Nombres de clases no definidos. Problema al cargar datos.")
        return

    model = get_model(NUM_CLASSES)
    model.load_state_dict(torch.load(MODEL_SAVE_PATH, map_location=device))
    model.eval()

    preprocess_webcam = data_transforms['test'] #

    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: No se pudo abrir la cámara web.")
        return

    print("Presiona 'q' para salir de la prueba con webcam.")
    font = cv2.FONT_HERSHEY_SIMPLEX

    while True:
        ret, frame = cap.read()
        if not ret:
            print("Error: No se pudo capturar el frame de la webcam.")
            break

        # OpenCV lee en BGR, PyTorch espera RGB
```

```
        img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)



        temp_preprocess = transforms.Compose([

            transforms.ToPILImage(),

        ] + preprocess_webcam.transforms)



        input_tensor = temp_preprocess(img_rgb)



        input_batch = input_tensor.unsqueeze(0)

        input_batch = input_batch.to(device)



        with torch.no_grad():

            output = model(input_batch)

            probabilities = torch.nn.functional.softmax(output[0],
dim=0)

            confidence, predicted_idx = torch.max(probabilities, 0)



            predicted_folder_name = class_names[predicted_idx.item()]

            predicted_sentiment =
SENTIMENT_MAP.get(predicted_folder_name, "Desconocido")



        text = f"{predicted_sentiment} ({confidence.item()*100:.1f}%)"

        cv2.putText(frame, text, (10, 30), font, 1, (0, 255, 0), 2,
cv2.LINE_AA)

        cv2.imshow('Clasificador de Emociones - Presiona Q para SALIR',
frame)
```

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

    cap.release()
    cv2.destroyAllWindows()
    print("Prueba con webcam finalizada.")

# --- 6. EJECUCIÓN PRINCIPAL ---
if __name__ == '__main__':
    # Entrenar el modelo
    #start_training()

    # Probar el modelo
    test_with_webcam()
```

7. Conclusiones y Posibles Mejoras

- El sistema propuesto utiliza una arquitectura ResNet18 con transfer learning para clasificar 7 emociones faciales.
- El rendimiento dependerá en gran medida de la calidad y cantidad de datos en el dataset imagenes_emo.
- **Posibles Mejoras:**
    - **Aumentar el Dataset:** Más imágenes y más variadas (diferentes personas, iluminaciones, ángulos) mejorarán la robustez.
    - **Fine-tuning Avanzado:** Descongelar más capas de ResNet18 y re-entrenarlas con una tasa de aprendizaje más baja.
    - **Probar otras Arquitecturas:** Modelos como EfficientNet o MobileNetV2 podrían ofrecer diferentes compromisos entre precisión y velocidad.
    - **Detección de Rostros:** Integrar un detector de rostros (ej. Haar Cascades, MTCNN, o modelos de Dlib/OpenCV DNN) para primero localizar el rostro en el frame de la webcam y luego pasar solo la región del rostro al clasificador de emociones. Esto haría el sistema más robusto a diferentes tamaños y posiciones de la cara.

- **Métricas de Evaluación Detalladas:** Calcular matriz de confusión, precisión, recall y F1-score por clase para entender mejor el rendimiento.

9. Video de pruebas del sistema.

https://youtu.be/z6nICPBv5CA