INSTITUTO TECNOLÓGICO DE CULIACÁN

INGENIERÍA EN SISTEMA COMPUTACIONALES

TAREA

Proyecto de Reconocimiento de Emociones en Tiempo Real

NOMBRE DE LA MATERIA:

Inteligencia Artificial

ALUMNOS:

Cruz Méndez Eymardh Sahid

Cardenas Quiñonez Angel

NOMBRE DEL DOCENTE:

Zuriel Dathan Mora Felix

Culiacán, Sin., 29 de Mayo de 2025

Tarea 2 Unidad 4 Proyecto de Reconocimiento de Emociones en Tiempo Real

Introducción

En este proyecto desarrollamos un sistema de reconocimiento de emociones humanas en tiempo real utilizando imágenes capturadas por una cámara. El sistema identifica emociones como enfado, desprecio, asco, miedo, felicidad, neutral, tristeza y sorpresa.

Instrucciones para ejecutar el proyecto

1. Prerrequisitos

- Python 3.8 o superior
- Pip (gestor de paquetes de Python)
- Cámara web (para pruebas en tiempo real)
- 8GB+ de RAM (recomendado para entrenamiento)

2. Instalación de Dependencias

```
# Crear entorno virtual (recomendado)
python -m venv emociones-env
source emociones-env/bin/activate  # Linux/Mac
.\emociones-env\Scripts\activate  # Windows

# Instalar dependencias principales
pip install torch torchvision torchaudio
pip install opencv-python numpy matplotlib scikit-learn pyyaml

# Para soporte GPU (opcional pero recomendado)
pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu118
```

3. Estructura del Proyecto

NOTA: para el caso de la carpeta data contine el dataset limpiado no se puede subir a GH por el tamaño del zip

enlace de descarga:

https://drive.google.com/file/d/11qlhyMdSVvxUjn719qbpJ90ktd5Fqy64/view?usp=sharing

Clonar o crear la siguiente estructura de directorios:

```
git clone
https://github.com/Eymardh/InteligenciaArtificialProyectos
```

```
Emotion-Recognition/
├── data/
│   ├── data.yaml
│   ├── train/
│   │   ├── images/
│   │   └── labels/
│   └── valid/
│       ├── images/
│       └── labels/
├── src/
│   ├── cameraTest.py
│   ├── clean_dataset.py
│   ├── detect.py
│   ├── model.py
│   ├── train.py
│   └── utils.py
├── models/
└── reports/
```

4. Preparación del Dataset

1. Descargar dataset de Roboflow en formato YOLO v8
2. Colocar las imágenes en:
    - `data/train/images/`
    - `data/valid/images/`
3. Colocar las anotaciones en:

    - `data/train/labels/`
    - `data/valid/labels/`
4. Verificar el archivo `data/data.yaml` contenga:

```
train: ../train/images
val: ../valid/images
nc: 8
names: ['Anger', 'Contempt', 'Disgust', 'Fear', 'Happy',
'Neutral', 'Sad', 'Surprise']
```

5. Limpieza del Dataset (Opcional pero recomendado)

```
cd src
python clean_dataset.py
```

6. Entrenamiento del Modelo

```
cd src
python train.py
```

Este proceso:

1. Entrenará el modelo por máximo 50 épocas
2. Guardará el mejor modelo en `models/best_model.pth`
3. Generará gráficos de entrenamiento en `reports/`

7. Probar con Cámara en Tiempo Real

```
cd src
python liveTest.py
```

Atajos durante la ejecución:

- `q`: Salir de la aplicación
- `d`: Activar/desactivar modo debug (muestra probabilidades)

8. Probar con Imágenes Estáticas

```
cd src
# Probar con imágenes de validación
```

```
python detect.py
```

```
# Probar con imágenes personalizadas
python detect.py --input ../custom_images/ --output
../runs/results/
```

9. Opciones Avanzadas

Entrenamiento con parámetros personalizados:

```
python train.py --batch_size 32 --lr 0.0001 --epochs 40
```

Detección en imagen específica:

```
python detect.py --image ../custom_images/mi_foto.jpg
```

Generar reporte de muestras del dataset:

```
python utils.py
```

Librerías principales utilizadas:

- **OpenCV**: Para procesamiento de imágenes en tiempo real y detección de rostros
- **PyTorch**: Framework de aprendizaje profundo para construir y entrenar nuestro modelo
- **NumPy**: Manejo eficiente de arrays y operaciones matemáticas
- **Scikit-learn**: Generación de reportes de clasificación
- **Matplotlib**: Visualización de resultados y distribución de datos
- **YAML**: Lectura de archivos de configuración

Dataset:

NOTA: Tomamos en cuenta la recomendación del profesor de usar un dataset mas completo y con imagenes de mejor calidad para facilitar el entrenamiento del modelo y obtener mejores resultados

enlace del dataset:
https://universe.roboflow.com/vietnamesegerman-university-mavjh/emotion-recognition-rjl9w/dataset/2
link de descarga:
https://drive.google.com/file/d/1Hz-ZT2lDLH1-WS0LYRkX8VlbZwdRVzHo/view?usp=sharing
Utilizamos el dataset **Emotion Recognition** de Roboflow en formato YOLO v8, que contiene:

- 8 clases de emociones: Enfado, Desprecio, Asco, Miedo, Felicidad, Neutral, Tristeza, Sorpresa
- 8,976 imágenes de entrenamiento
- 2,244 imágenes de validación
- Estructura YOLO con archivos de imágenes y anotaciones en formato TXT

2. Explicación de los Archivos del Proyecto

`model.py`

Función: Define la arquitectura de la red neuronal convolucional (CNN) para clasificación de emociones.

Características clave:

- **Arquitectura CNN**: 4 bloques convolucionales con BatchNorm y MaxPooling
- **Capas completamente conectadas**: 2 capas con Dropout para regularización
- **Diseño ligero**: Optimizado para inferencia en tiempo real
- **Salida**: 8 neuronas (una por cada emoción) con activación softmax

Mejoras implementadas:

```
class ModelCNN(nn.Module):
    def __init__(self, num_classes=8):
```

```
        super(ModelCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        self.conv4 = nn.Conv2d(256, 512, kernel_size=3, padding=1)
        self.bn4 = nn.BatchNorm2d(512)

        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.5)

        self.fc1 = nn.Linear(512 * 6 * 6, 1024)  # Ajustamos el
tamaño de entrada
        self.fc2 = nn.Linear(1024, num_classes)

        def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))  # 64x48x48
        x = self.pool(F.relu(self.bn2(self.conv2(x))))  # 128x24x24
        x = self.pool(F.relu(self.bn3(self.conv3(x))))  # 256x12x12
        x = self.pool(F.relu(self.bn4(self.conv4(x))))  # 512x6x6

        x = x.view(-1, 512 * 6 * 6)
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.fc2(x)
        return x
```

`train.py`

Función: Entrenamiento del modelo de reconocimiento de emociones.

Proceso clave:

1. **Carga de datos**: Utiliza `YOLOEmotionDataset` para cargar imágenes y etiquetas
2. **Balanceo de clases**: Implementa muestreo ponderado (`WeightedRandomSampler`)
3. **Configuración de entrenamiento**:
    - Optimizador Adam con tasa de aprendizaje reducida (0.0001)
    - Función de pérdida con ponderación por clases
    - Scheduler `ReduceLROnPlateau` para ajuste dinámico del learning rate

4. **Entrenamiento**:
    - 50 épocas máximas con parada temprana
    - Validación después de cada época
    - Guardado del mejor modelo
5. **Visualización**: Genera gráficos de pérdida y precisión

Características destacables:

```
  sample_weights = [class_weights[label] for _, label in train_dataset]
   sampler = WeightedRandomSampler(
       weights=sample_weights,
       num_samples=len(sample_weights),
       replacement=True
   )
```

```
   # Learning rate scheduler para reducir el LR en caso de no mejora
scheduler = optim.lr_scheduler.ReduceLROnPlateau(
       optimizer,
       mode='max',
       factor=0.5,
       patience=2
   )
```

utils.py

Función: Utilidades para manejo de datos y transformaciones.

Componentes principales:

- **YOLOEmotionDataset**:
    - Carga imágenes y etiquetas en formato YOLO
    - Realiza crop de rostros usando anotaciones
    - Aplica padding y redimensionamiento a 96x96
    - Manejo de errores y casos extremos
- **Transformaciones**:
    - Aumento de datos para entrenamiento (rotaciones, flips, cambios de contraste)
    - Transformaciones básicas para validación

Implementación clave:

```
   def __getitem__(self, idx):
       img_name = self.image_files[idx]
       img_path = os.path.join(self.image_dir, img_name)
```

```
        # Leemos y convertimos la imagen a escala de grises
        try:
            img = cv2.imread(img_path)
            if img is None:
                raise ValueError(f"Could not read image {img_path}")
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        except Exception as e:
            print(f"Error reading image {img_path}: {e}")
            # Retornamos una imagen en blanco como placeholder
            # Esto es útil para evitar que el entrenamiento falle por
una imagen corrupta
            gray = np.zeros((96, 96), dtype=np.uint8)
            return self.transform(gray) if self.transform else gray, 0
```

liveTest.py

Función: Detección de emociones en tiempo real usando cámara web.

Flujo de trabajo:

1. Inicialización del modelo y detector de rostros (Haar Cascade)
2. Captura de frames de la cámara
3. Detección de rostros en cada frame
4. Preprocesamiento de rostros (CLAHE para mejora de contraste)
5. Inferencia del modelo
6. Visualización de resultados con etiquetas en español
7. Modo debug para mostrar probabilidades

Características innovadoras:

```
# Mapeamos las emociones a español

        self.spanish_labels = {

            'Anger': 'Enojado',

            'Contempt': 'Molesto',

            'Disgust': 'Asco',

            'Fear': 'Miedo',

            'Happy': 'Feliz',

            'Neutral': 'Neutral',

            'Sad': 'Triste',
```

```
            'Surprise': 'Sorpresa'
        }

        # TRansformaciones de imagen
        self.transform = transforms.Compose([
            transforms.ToPILImage(),
            transforms.Resize((96, 96)),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.5], std=[0.5])
        ])

        # Cargar el clasificador Haar para detección de rostros
        self.face_cascade = cv2.CascadeClassifier(
            cv2.data.haarcascades +
'haarcascade_frontalface_default.xml'
        )

        # Creamos el objeto CLAHE para mejorar el contraste
        self.clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))

        self.debug = False
```

`detect.py`

Función: Detección de emociones en imágenes estáticas.

Diferencias con `liveTest.py`:

- Procesamiento de imágenes individuales o directorios completos

- Guardado de resultados en lugar de visualización en tiempo real
- Sin funcionalidad de cámara en vivo

`clean_dataset.py`

Función: Limpieza y preparación del dataset.

Operaciones principales:

- Eliminación de archivos de etiquetas vacíos
- Eliminación de imágenes sin rostros detectables
- Filtrado de rostros muy pequeños
- Eliminación de imágenes corruptas

Conclusiones y Aprendizajes

Al desarrollar este proyecto de reconocimiento de emociones, enfrentamos varios desafíos técnicos y conceptuales que nos permitieron crecer como equipo en el campo de visión por computadora y aprendizaje profundo.

Uno de los primeros retos fue comprender la complejidad del reconocimiento de emociones, una tarea que incluso para humanos puede ser subjetiva. Nos sorprendió descubrir cómo factores como la iluminación, la orientación del rostro y las expresiones sutiles podrían afectar significativamente el rendimiento del modelo.

Durante el proceso de entrenamiento, aprendimos la importancia crítica del procesamiento de datos. Implementar técnicas como CLAHE para mejorar el contraste y el balance de clases mediante muestreo ponderado resultó fundamental para mejorar la precisión de nuestro modelo. También descubrimos que el aumento de datos no es simplemente una técnica opcional, sino una necesidad para crear modelos robustos.

La implementación en tiempo real nos enseñó lecciones valiosas sobre optimización. Tuvimos que encontrar un equilibrio entre la complejidad del modelo y la velocidad de inferencia, optando finalmente por una arquitectura que mantuviera una buena precisión sin comprometer el rendimiento en tiempo real.

Uno de los aprendizajes más significativos fue comprender que un buen modelo no depende únicamente de la arquitectura neuronal, sino de todo el pipeline de procesamiento. Desde la detección precisa de rostros hasta la normalización de las imágenes, cada etapa demostró ser crítica para el resultado final.

El desarrollo de las utilidades de visualización y debug, especialmente el modo de probabilidades en `liveTest.py`, nos permitió entender mejor el comportamiento del modelo y diagnosticar problemas cuando las predicciones no eran las esperadas.

Finalmente, trabajar con un dataset real nos enseñó sobre la importancia de la limpieza y verificación de datos. El script `clean_dataset.py` surgió de la necesidad de eliminar ejemplos problemáticos que afectan negativamente el entrenamiento.

En conclusión, este proyecto no solo nos permitió aplicar conceptos teóricos de aprendizaje profundo, sino también desarrollar habilidades prácticas en solución de problemas complejos, trabajo en equipo y optimización de sistemas. El resultado final, capaz de reconocer emociones humanas en tiempo real, representa una síntesis valiosa de todos estos aprendizajes y un punto de partida para futuras mejoras e investigaciones.