Modelo Light CNN-9 para reconocimiento de emociones

Entrenamiento, Evaluación y Ejecución

Dr. Hugo Arnoldo Mitre Hernández

CIMAT Zacatecas

Marzo 2026

Dataset de expresiones faciales

Para el entrenamiento del modelo se utilizó el conjunto de datos **Radboud Faces Database (RaFD)**.
Este dataset contiene imágenes de rostros humanos con diferentes expresiones emocionales capturadas en condiciones controladas de iluminación y posición.

Las clases consideradas en este estudio fueron:

- enojado
- feliz
- miedo
- neutral
- sorpresa
- triste

Las imágenes originales tienen una resolución aproximada de 681 x 1024 píxeles.
Cada carpeta del dataset corresponde a una clase emocional utilizada para el entrenamiento supervisado del modelo.

Preprocesamiento de las imágenes

Antes del entrenamiento del modelo se aplicaron varias transformaciones para estandarizar las entradas de la red neuronal.

- Redimensionamiento de las imágenes a 128×128 píxeles
- Conversión a escala de grises
- Normalización de intensidades en el rango $[0, 1]$

Además, se aplicaron técnicas de *data augmentation* para mejorar la capacidad de generalización del modelo:

- rotación aleatoria
- zoom aleatorio
- volteo horizontal

Estas transformaciones permiten generar variaciones de las imágenes durante el entrenamiento y reducir el riesgo de sobreajuste.

Modelo LightCNN-9

El modelo utilizado es LightCNN-9, una arquitectura convolucional diseñada para tareas de análisis facial.

La red emplea una función especial denominada *Max Feature Map (MFM)* definida como:

$$MFM(x_1, x_2) = \text{máx}(x_1, x_2)$$

Esta operación selecciona el mapa de características más informativo entre dos filtros convolucionales.

La arquitectura del modelo incluye:

- capas convolucionales con activación MFM
- operaciones de *max pooling*
- capas densas finales
- capa *softmax* para clasificación en seis clases

El entrenamiento se realiza mediante la función de pérdida *cross entropy* y el optimizador Adam.

Evaluación del modelo

El rendimiento del modelo se evaluó utilizando métricas estándar de clasificación multiclase.

- Accuracy
- F1-score
- Matriz de confusión

La métrica de exactitud se define como:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

El F1-score combina precisión y recall:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

La matriz de confusión permite analizar el desempeño del modelo en cada una de las clases emocionales.

Archivo 1: train_lightcnn_rafd.py – Configuración

```
import tensorflow as tf                # Biblioteca de aprendizaje profundo
from tensorflow.keras import layers, models  # Capas y modelos de Keras
import pathlib                         # Manejo de rutas de archivos

# CONFIGURACION: Tamano de imagen y lotes
img_size = (128, 128)     # Redimensionamos a 128x128 pixeles
batch_size = 32           # Procesamos 32 imagenes a la vez
```

¿Qué hace? Importa las herramientas necesarias y define el tamaño de las imágenes (128×128 píxeles en escala de grises) y cuántas imágenes procesar por lote (32).

Archivo 1: train_lightcnn_rafd.py – Preprocesamiento

Listing 1: Normalización y aumento de datos

```
# Normalizacion: convierte pixeles de [0,255] a [0,1]
normalization_layer = layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))

# Aumento de datos: crea variaciones artificiales
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),    # Voltea horizontalmente
    layers.RandomRotation(0.05),        # Rota +-5%
    layers.RandomZoom(0.1)              # Zoom +-10%
])
```

¿Qué hace? Normalización: divide cada pixel por 255 para tener valores entre 0 y 1. **Aumento de datos:** genera versiones ligeramente modificadas de las imágenes para que el modelo aprenda mejor.

Archivo 1: train_lightcnn_rafd.py – Capa MFM (El Corazón)

Listing 2: Max Feature Map: la innovacion de LightCNN

```
class MFM(layers.Layer):
    def __init__(self, filters, kernel_size, ...):
        # Crea el doble de filtros (ej: 48 -> 96)
        self.conv = layers.Conv2D(filters*2, kernel_size, ...)

    def call(self, x):
        x = self.conv(x)            # Aplica convolucion
        x1, x2 = tf.split(x, 2, -1) # Divide en 2 mitades
        return tf.maximum(x1, x2)   # Toma el maximo
```

¿Qué hace? MFM es la clave de LightCNN. En lugar de usar ReLU, crea el doble de filtros, los divide en 2 y toma el máximo. Esto reduce parámetros y mejora la selección de características.

Archivo 1: train_lightcnn_rafd.py – Arquitectura Completa

Listing 3: Construccion de la red neuronal

```
inputs = layers.Input(shape=(128,128,1))  # Entrada: imagen 128x128 gris

x = data_augmentation(inputs)    # Paso 1: Aumento de datos

# Paso 2: Bloques convolucionales con MFM y pooling
x = MFM(48, 5)(x);    x = layers.MaxPool2D(2)(x)    # 128->64
x = MFM(96, 3)(x);    x = layers.MaxPool2D(2)(x)    # 64->32
x = MFM(192, 3)(x);   x = layers.MaxPool2D(2)(x)    # 32->16
x = MFM(128, 3)(x);   x = layers.MaxPool2D(2)(x)    # 16->8

x = layers.Flatten()(x)          # Paso 3: Aplana a vector
x = MFMDense(256)(x)             # Paso 4: Capa densa con MFM
x = layers.Dropout(0.5)(x)       # Paso 5: Apaga 50% neuronas (evita sobreajuste)
outputs = layers.Dense(6, activation="softmax")(x)  # Paso 6: 6 clases de salida
```

¿Qué hace? Construye la red: 4 bloques convolucionales que reducen la imagen de 128→8 píxeles, luego capas densas para clasificar en 6 emociones.

Archivo 1: train_lightcnn_rafd.py – Entrenamiento y Guardado

Listing 4: Compilacion, entrenamiento y guardado

```
# Configura como aprender
model.compile(
    optimizer="adam",                    # Algoritmo de optimizacion
    loss="sparse_categorical_crossentropy",  # Funcion de error
    metrics=["accuracy"]                 # Metrica: precision
)

# Entrena por 25 epocas (pasa 25 veces por todos los datos)
history = model.fit(train_ds, epochs=25)

# Guarda el modelo entrenado para usarlo despues
model.save("lightcnn9_rafd.h5")
```

¿Qué hace? Compile: configura el cómo.ªprender (Adam ajusta pesos, crossentropy mide error). **Fit:** entrena 25 épocas. **Save:** guarda el modelo en disco.

Problema: La red neuronal devuelve 6 números (logits), pero necesitamos **probabilidades** que sumen 1.

Fórmula Softmax:

$$\sigma(z_i) = \frac{e^{z_i}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4} + e^{z_5} + e^{z_6}}$$

Ejemplo visual:

	Entrada	$\rightarrow$	**Cálculo**	$\rightarrow$	**Salida**	
Enojado	0.5		$e^{0,5} = 1,65$		0.09	(9 %)
Feliz	**2.0**		$e^{2,0} = 7,39$		**0.50**	**(50 %)**
Miedo	-1.0		$e^{-1,0} = 0,37$		0.02	(2 %)
Neutral	0.0		$e^{0,0} = 1,00$		0.07	(7 %)
Sorpresa	1.0		$e^{1,0} = 2,72$		0.18	(18 %)
Triste	-0.5		$e^{-0,5} = 0,61$		0.04	(4 %)
				Suma	**1.00**	**(100 %)**

La clase con mayor logit (Feliz = 2.0) obtiene la mayor probabilidad (50 %).

Características:

1. **Normalización:** Todas las probabilidades suman exactamente 1.
2. **Positividad:** Todas las salidas son > 0 (gracias a e^x).
3. **Amplificación:** Diferencias pequeñas en logits $\rightarrow$ grandes diferencias en probabilidades.
4. **Diferenciable:** Se puede derivar para entrenar con gradient descent.

Decisión final:

$$\text{Clase predicha} = \arg\max(p_1, p_2, p_3, p_4, p_5, p_6)$$

Es decir: el índice con mayor probabilidad.

Comparación: Logits vs Probabilidades

Emoción	Logit	Prob.
Enojado	0.5	9 %
Feliz	**2.0**	**50 %**
Miedo	-1.0	2 %
Neutral	0.0	7 %
Sorpresa	1.0	18 %
Triste	-0.5	4 %

Observación: Feliz tiene solo 2x el logit de Sorpresa (2.0 vs 1.0), pero 2.8x la probabilidad (50 % vs 18 %).

Softmax "exagera" las diferencias.

Archivo 2: evaluate_lightcnn.py – Carga y Preparación

Listing 5: Importaciones y definicion de capas

```
import tensorflow as tf
import numpy as np
import pathlib
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report, ...

# DEFINIR CAPAS PERSONALIZADAS (mismas que en entrenamiento)
class MFM(tf.keras.layers.Layer): ...    # Igual que antes
class MFMDense(tf.keras.layers.Layer): ...  # Igual que antes
```

¿Qué hace? Importa herramientas para evaluar. **Importante:** debe redefinir las capas MFM porque al cargar el modelo guardado, TensorFlow necesita saber qué son.

Archivo 2: evaluate_lightcnn.py – Carga del Modelo

Listing 6: Carga del modelo entrenado

```
# Diccionario con las capas personalizadas
custom_objects = {'MFM': MFM, 'MFMDense': MFMDense}

# Carga el modelo usando las capas definidas
with tf.keras.utils.custom_object_scope(custom_objects):
    model = tf.keras.models.load_model("lightcnn9_rafd.h5", compile=False)

print("Modelo cargado exitosamente")
```

¿Qué hace? Carga el archivo `.h5` guardado. El `custom_object_scope` le dice a TensorFlow üsa estas definiciones de MFM y MFMDense cuando encuentres esas capas en el archivo".

Archivo 2: evaluate_lightcnn.py – Predicciones

Listing 7: Obtener predicciones del modelo

```
y_true = []     # Lista para guardar etiquetas reales
y_pred = []     # Lista para guardar predicciones del modelo

# Itera por todos los lotes del dataset
for images, labels in test_ds:
    preds = model.predict(images, verbose=0)  # Predice probabilidades
    preds = np.argmax(preds, axis=1)          # Toma la clase con mayor prob.

    y_pred.extend(preds)           # Guarda predicciones
    y_true.extend(labels.numpy()) # Guarda etiquetas reales

y_true = np.array(y_true)  # Convierte a arrays de NumPy
y_pred = np.array(y_pred)
```

¿Qué hace? Pasa todas las imágenes por el modelo, obtiene predicciones (6 números = probabilidades por clase), toma el índice del máximo (la clase predicha), y guarda tanto predicciones como valores reales para comparar.

Archivo 2: evaluate_lightcnn.py – Métricas

Listing 8: Calculo de metricas de rendimiento

```python
# Precision global: aciertos / total
acc = accuracy_score(y_true, y_pred)
print(f"Accuracy: {acc:.4f}")  # Ej: 0.8534 = 85.34%

# F1-score: balance entre precision y recall
f1 = f1_score(y_true, y_pred, average="weighted")
print(f"F1-score: {f1:.4f}")

# Reporte detallado por clase
print(classification_report(y_true, y_pred, target_names=class_names))
# Muestra: precision, recall, f1-score para cada emocion
```

¿Qué hace? Calcula métricas: **Accuracy** (% de aciertos totales), **F1-score** (balance entre precisión y sensibilidad), y **reporte por clase** (cómo le fue a cada emoción individualmente).

Archivo 2: evaluate_lightcnn.py – Matriz de Confusión

Listing 9: Matriz de confusion y visualizacion

```
# Crea matriz: filas=real, columnas=prediccion
cm = confusion_matrix(y_true, y_pred)

# Visualizacion con matplotlib
plt.figure(figsize=(10, 8))
plt.imshow(cm, interpolation="nearest", cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.colorbar()
plt.xticks(tick_marks, class_names, rotation=45)
plt.yticks(tick_marks, class_names)

# Escribe el numero en cada celda
for i in range(len(class_names)):
    for j in range(len(class_names)):
        plt.text(j, i, cm[i,j], ha="center", va="center")

plt.savefig('confusion_matrix.png')  # Guarda imagen
plt.show()                           # Muestra en pantalla
```

Archivo 3: realtime_emotion.py – Configuración

Listing 10: Importaciones y definicion de capas (igual que antes)

```
import cv2                                    # OpenCV: vision por computadora
import tensorflow as tf
import numpy as np

# DEFINIR CAPAS PERSONALIZADAS (MFM y MFMDense)
class MFM(tf.keras.layers.Layer): ...
class MFMDense(tf.keras.layers.Layer): ...

# Cargar modelo entrenado
custom_objects = {'MFM': MFM, 'MFMDense': MFMDense}
with tf.keras.utils.custom_object_scope(custom_objects):
    model = tf.keras.models.load_model("lightcnn9_rafd.h5", compile=False)

class_names = ["enojado", "feliz", "miedo", "neutral", "sorpresa", "triste"]
```

¿Qué hace? Importa OpenCV (cv2) para acceder a la cámara. Carga el mismo modelo entrenado. Define los nombres de las 6 emociones para mostrarlas en pantalla.

Archivo 3: realtime_emotion.py – Detector de Rostros

Listing 11: Inicializacion del detector de rostros

```
# Carga el clasificador Haar Cascade (incluido en OpenCV)
face_cascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
)

# Abre la camara (0 = camara por defecto)
cap = cv2.VideoCapture(0)

print("Presiona 'q' para salir")
```

¿Qué hace? Carga un detector de rostros pre-entrenado (Haar Cascade) que viene con OpenCV. Abre la cámara web. El archivo `.xml` contiene patrones para detectar rostros frontales.

Archivo 3: realtime_emotion.py – Bucle Principal

Listing 12: Captura y procesamiento de video

```
while True:
    ret, frame = cap.read()            # Lee un frame de la camara
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # Convierte a grises

    # Detecta rostros en la imagen
    faces = face_cascade.detectMultiScale(
        gray,
        scaleFactor=1.3,    # Reduccion de escala entre busquedas
        minNeighbors=5      # Minimo de vecinos para confirmar rostro
    )
```

¿Qué hace? Bucle infinito que: 1) Captura frame de la cámara, 2) Convierte a escala de grises (el modelo fue entrenado así), 3) Busca rostros con el detector Haar.

Archivo 3: realtime_emotion.py – Predicción y Dibujo

Listing 13: Procesamiento de cada rostro detectado

```
for (x, y, w, h) in faces:              # Para cada rostro encontrado
    face = gray[y:y+h, x:x+w]           # Recorta la region del rostro

    face = cv2.resize(face, (128, 128))    # Redimensiona a 128x128
    face = face / 255.0                    # Normaliza a [0,1]
    face = np.reshape(face, (1, 128, 128, 1))  # Formato: (batch, h, w, canales)

    prediction = model.predict(face, verbose=0)  # Predice emocion
    emotion_index = np.argmax(prediction)          # Clase con mayor prob.
    emotion = class_names[emotion_index]           # Nombre de la emocion
    confidence = np.max(prediction)              # Probabilidad (0-1)
```

¿Qué hace? Para cada rostro: recorta, prepara igual que en entrenamiento (128×128, normalizado), predice con el modelo, obtiene la emoción y su confianza.

Archivo 3: realtime_emotion.py – Visualización

Listing 14: Dibujar resultados en pantalla

```
    # Crea etiqueta: "feliz (0.92)"
    label = f"{emotion} ({confidence:.2f})"

    # Dibuja rectangulo verde alrededor del rostro
    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    # Escribe la emocion arriba del rectangulo
    cv2.putText(frame, label, (x, y-10),
                cv2.FONT_HERSHEY_SIMPLEX,  # Tipo de letra
                0.9,                       # Tamaño
                (0, 255, 0),               # Color verde (BGR)
                2)                         # Grosor

cv2.imshow("Emotion Recognition", frame)  # Muestra en ventana

if cv2.waitKey(1) & 0xFF == ord("q"):     # Si presionas 'q', sale
    break
```

¿Qué hace? Dibuja un rectángulo verde alrededor del rostro y muestra la emoción predicha

Archivo 3: realtime_emotion.py – Limpieza

Listing 15: Liberar recursos al terminar

```
cap.release()             # Libera la camara (apaga)
cv2.destroyAllWindows()   # Cierra todas las ventanas de OpenCV
```

¿Qué hace? Libera la cámara para que otros programas puedan usarla y cierra las ventanas.
Importante: si no se hace, la cámara queda "bloqueada"hasta reiniciar.

Resumen: Flujo Completo del Sistema

Orden	Archivo	Función
1	`train_lightcnn_rafd.py`	**Entrena** el modelo con imágenes etiquetadas. Crea `lightcnn9_rafd.h5`
2	`evaluate_lightcnn.py`	**Evalúa** qué tan bueno es el modelo. Genera métricas y matriz de confusión
3	`realtime_emotion.py`	**Usa** el modelo en vivo con la cámara web

Dependencias: Los 3 archivos necesitan las mismas definiciones de `MFM` y `MFMDense` para que el modelo funcione correctamente.

Gracias