TECNOLÓGICO
NACIONAL DE MÉXICO

UNIDAD IV – Aplicación de Detección Facial

Alumnos:

- Jorge Alberto Lara Beltrán
- Diego Pacheco Pérez

Docente:

Zuriel Dathan Mora Felix

Materia:

Inteligencia Artificial

A. Arquitectura del modelo

- Se eligió una Red Neuronal Convolucional (CNN) porque es la arquitectura más adecuada para tareas de visión por computadora.
- Las CNN son capaces de extraer características espaciales de las imágenes (bordes, texturas, patrones) y combinarlas en niveles más altos de abstracción.
- Para detección facial y reconocimiento emocional, esta arquitectura es ideal porque:
    - Reduce la complejidad de los datos de entrada (imágenes).
    - Generaliza bien en variaciones de iluminación, rotación y expresiones.
    - Permite trabajar con imágenes en escala de grises (48x48 píxeles), optimizando el rendimiento.

B. Tipo de red neuronal

- Se construyó un modelo personalizado desde cero en lugar de usar un modelo preentrenado como VGG16 o ResNet.
- Ventajas del modelo personalizado:
    - Adaptado específicamente al tamaño de las imágenes (48x48).
    - Menor consumo de recursos que un modelo preentrenado.
    - Flexibilidad para ajustar hiperparámetros y capas.
- Desventajas:
    - Requiere tiempo de entrenamiento.
    - Puede tener menor precisión inicial comparado con modelos preentrenados.

C. Herramientas y Tecnologías

- Lenguaje: Python, versátil y con gran soporte en IA.
- Bibliotecas:
    - TensorFlow/Keras: para construir y entrenar la red neuronal.
    - OpenCV: para detección de rostros en tiempo real con la cámara.
    - NumPy: para manipulación de matrices e imágenes.
- Cada herramienta cumple un rol:
    - Keras simplifica la construcción de modelos.
    - OpenCV permite la integración con la webcam y detección facial.
    - NumPy gestiona el preprocesamiento de datos.

D. Conjuntos de datos

- Se recomienda usar FER2013 o AffectNet, que contienen miles de imágenes etiquetadas con emociones, en este caso elegimos FER2013.
- Preprocesamiento aplicado:

    - Escalado de valores de píxeles a rango [0,1].
    - Conversión a escala de grises.
    - Aumento de datos (rotación, brillo, etc.) para mejorar la generalización.
- Esto asegura que el modelo no se sobreajuste y pueda reconocer emociones en condiciones variadas.

A2. Descripción del modelo

- El modelo es una CNN diseñada para clasificar emociones faciales en 7 categorías: angry, disgusted, fearful, happy, neutral, sad, surprised.
- Funciona tomando imágenes de rostros detectados en tiempo real, preprocesándolos y clasificándolos con la red neuronal entrenada.
- Resultados esperados: precisión cercana al 65–75% en validación, dependiendo del dataset y número de épocas.

B2. Fases de construcción

1. **Recogida de datos:**
    - Se seleccionó un dataset FER2013
    - Se aplicó preprocesamiento: normalización, escala de grises, aumento de datos.
    - Reto: trabajar con pocas imágenes de “disgusted”
2. **Diseño del modelo:**
    - Se definió una CNN con varias capas convolucionales, normalización y dropout para evitar sobreajuste.
    - Reto: ajustar número de filtros y tamaño de kernel para optimizar rendimiento.
3. **Entrenamiento:**
    - Se entrenó con Adam como optimizador y categorical_crossentropy como función de pérdida.
    - Se usó EarlyStopping para detener el entrenamiento si no mejoraba la validación.
    - Reto: encontrar el número óptimo de épocas (20–30).
4. **Pruebas y validación:**
    - Se evaluó el modelo con el conjunto de validación.
    - Se probó en tiempo real con la webcam.
    - Reto: mantener precisión en condiciones de iluminación variable y rostros parciales.

Código

1. cargar_datos()

- **Propósito:** Prepara y carga los datos de entrenamiento y validación desde carpetas locales.
- **Qué hace:**
  - Usa `ImageDataGenerator` para aplicar preprocesamiento y aumentación de datos (rotación, brillo, normalización).
  - Genera lotes de imágenes en escala de grises de tamaño 48x48.
  - Devuelve dos generadores: uno para entrenamiento y otro para validación.
- **Importancia:** Permite que el modelo aprenda con datos variados y evita el sobreajuste.

2. construir_modelo(input_s, num_clases)

- **Propósito:** Define la arquitectura de la red neuronal convolucional (CNN).
- **Qué hace:**
  - Crea un modelo `Sequential` con capas convolucionales, normalización, pooling y dropout.
  - Incluye capas densas finales con regularización L2 para mejorar la generalización.
  - Compila el modelo con el optimizador Adam y la función de pérdida `categorical_crossentropy`.
- **Importancia:** Es el núcleo del sistema, donde se define cómo se procesan las imágenes y cómo se clasifican las emociones.

3. entrenar_modelo(model, train_gen, test_gen, epochs=20)

- **Propósito:** Entrena el modelo con los datos de entrenamiento y validación.
- **Qué hace:**
  - Usa `EarlyStopping` para detener el entrenamiento si la precisión de validación deja de mejorar.
  - Ajusta los pesos de la red durante varias épocas.
  - Devuelve el modelo entrenado.
- **Importancia:** Optimiza el rendimiento del modelo y evita entrenamientos innecesarios que podrían llevar al sobreajuste.

4. guardar_modelo(model, ruta='model/modelo_emociones.h5')

- **Propósito:** Guarda el modelo entrenado en un archivo `.h5`.
- **Qué hace:**

    - Utiliza `model.save()` para almacenar la arquitectura y los pesos.
- **Importancia:** Permite reutilizar el modelo sin necesidad de volver a entrenarlo.

5. usar_modelo_webcam(ruta_modelo='model/modelo_emociones.h5')

- **Propósito:** Usa el modelo entrenado para detectar emociones en tiempo real con la cámara.
- **Qué hace:**
    - Carga el modelo guardado.
    - Usa `CascadeClassifier` de OpenCV para detectar rostros en el video.
    - Preprocesa cada rostro detectado (escala de grises, redimensionado, normalización).
    - Predice la emoción con el modelo y muestra el resultado en pantalla con un rectángulo y texto.
    - Finaliza cuando se presiona la tecla `q`.
- **Importancia:** Es la aplicación práctica que conecta el modelo con la interacción en tiempo real.

6. Bloque final del script

```
train_gen, test_gen = cargar_datos()
model = load_model('model/modelo_emociones.h5')
model.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
usar_modelo_webcam('model/modelo_emociones.h5')
```

- **Propósito:** Ejecuta el flujo completo.
- **Qué hace:**
    - Carga los datos.
    - Carga el modelo previamente entrenado.
    - Compila el modelo para asegurar que esté listo para predicciones.
    - Lanza la aplicación de detección de emociones en la webcam.
- **Importancia:** Es el punto de entrada que conecta todas las funciones y ejecuta la aplicación.