TECNOLÓGICO NACIONAL DE MÉXICO

INTELIGENCIA ARTIFICIAL

UNIDAD 4 – TAREA 2

NOMBRE:

SARABIA GUZMÁN JESÚS ALDHAIR

ZAMUDIO LIZÁRRAGA BRYAN MARTÍN

NUMERO DE CONTROL:

22170824

22170855

FECHA:

29/05/2025

MAESTRO:

ZURIEL DATHAN MORA FÉLIX

Para crear esta red neuronal tuvimos que investigar sobre cómo funciona tensorflow/keras (los métodos que tiene, más o menos cuáles son los mejores parámetros para usar al entrenar un modelo como el nuestro, cómo funciona el tema de las capas y de compilar el modelo al final, etc.) a través de las documentaciones, videos, artículos, etc. También hubo bastante experimentación de nuestra parte para encontrar unos parámetros que nos dieran los mejores resultados posibles, entrenando una y otra vez con el dataset mencionado en la tarea anterior FER-2013, con diferentes configuraciones y guardando los mejores resultados.

Pre-procesamiento

Intentamos varios parámetros tanto en el preprocesado como en el modelo en sí partiendo de los que usamos para la tarea pasada para el preprocesado, hasta llegar a estos que fueron los que nos dieron mejores resultados:

```
datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.15,
    brightness_range=[0.9, 1.1],
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2
)
```

Para el generador en sí también cambiamos de ‘rgb’ a ‘grayscale’, ya que las imágenes son en blanco y negro

```
train_generator = datagen.flow_from_directory(
    ruta,
    target_size=(48, 48),
    color_mode='grayscale',
    batch_size=32,
    class_mode='categorical',
    subset='training'
)
```

Red Neuronal

Para la red neuronal, decidimos hacerlo Sequential, ya que en teoría según la documentación de keras es lo mejor cuando tienes un flujo de datos “lineal”, una entrada y una salida, como es nuestro caso. Creamos varias capas con Conv2D, empezando con 32 filtros de 3x3, duplicando los filtros con cada capa y aplicando el pooling de 2x2 entre cada capa, usando BatchNormalization para acelelar el procesa, además de la activation=’relu’, una activación estándar en estos casos.

Al final aplanamos para la última capa densa, aplicamos un dropout de 40% para ayudar a evitar el sobreajuste y usamos la activación softmax para clasificar la entrada en una de las 5 emociones que tenemos.

```
# Modelo, red neuronal
modelo = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(96, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(256, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.4),
    Dense(train_generator.num_classes, activation='softmax')
])
```

Antes de llegar a esta configuración intentamos otras, como ir aumentando los filtros de 32 en 32 en cada capa, o diferentes números de capas, usar diferente valor para el dropout, o usar dropout entre capa y capa, sin embargo, esta configuración fue la que

mejores resultados nos arrojó, tanto en precisión, reducción de tiempo de entreno, reducción de sobreajuste, etc.

Por último, entrenamos con un máximo de 50 epochs, y usamos callbacks con EarlyStopping para asegurarnos que se detenga cuando se estanque el entrenamiento para que no haya sobreajuste, y se vaya guardando el mejor modelo entrenado en el archivo mejor_modelo.keras.

Aplicación del modelo

Usamos OpenCV para capturar video en tiempo real desde la cámara y para detectar caras en cada fotograma mediante un clasificador ya preentrenado (haarcascade_frontalface_default.xml), para poder enfocarnos en la cara.

Cada vez que se detecta un rostro:

- Lo recortamos y lo convertimos a escala de grises.
- Lo redimensionamos a 48×48 píxeles, el tamaño de entrada esperado por el modelo.
- Normalizamos los valores de píxeles a un rango entre 0 y 1.
- Ajustamos la forma del array con np.expand_dims para que sea compatible con la entrada del modelo (batch size, alto, ancho, canales).

Luego usamos el modelo para hacer la predicción. Con np.argmax obtenemos la clase con mayor probabilidad, y verificamos la confianza de la predicción. Si la probabilidad supera el umbral del 50%, se muestra la etiqueta escogida con su porcentaje. Si no, se imprime que la predicción es insegura.