Instituto tecnológico de Culiacán

Nombres:

Carrasco Medina Carlos Ivan

Peña Vizcarra Jatniel Alejandro

Materia:

Inteligencia artificial

Maestro:

Mora Felix Zuriel Dathan

Trabajo:

Tarea 2 unidad 4

Fecha:

29/05/2025

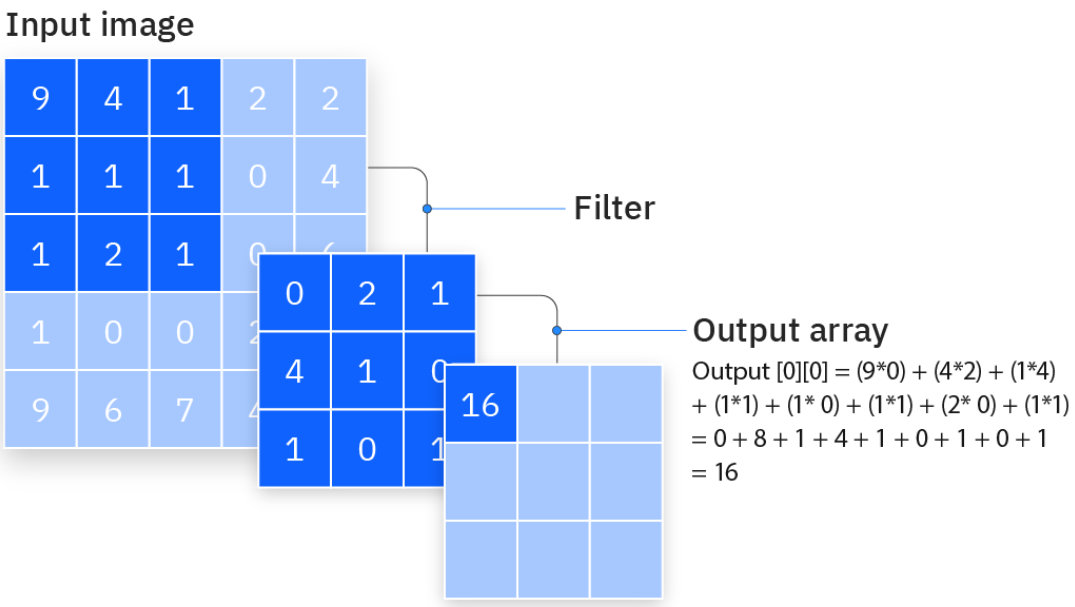
neuronal **convolucional secuencial (CNN):**

Las CNN se distinguen de otras redes neuronales por su mejor desempeño con entradas de señal de imagen, voz o audio. Tienen tres tipos principales de capas, que son:

- Capa convolucional
- Capa de agrupamiento
- Capa totalmente conectada (FC)

La capa convolucional es la primera capa de una red convolucional. Si bien a las capas convolucionales las pueden seguir capas convolucionales adicionales o capas agrupadas, la capa totalmente conectada es la capa final. Con cada capa, la CNN aumenta su complejidad, identificando mayores porciones de la imagen. Las primeras capas se enfocan en características simples, como colores y bordes. A medida que los datos de la imagen avanzan a través de las capas de CNN, se comienzan a reconocer elementos o formas más grandes del objeto, hasta que finalmente se identifica el objeto previsto.

Ejemplo:



Modelo en el programa:

```
model = tf.keras.Sequential([
  tf.keras.layers.Rescaling(1./255, input_shape=(224, 224, 3)),
  tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
```

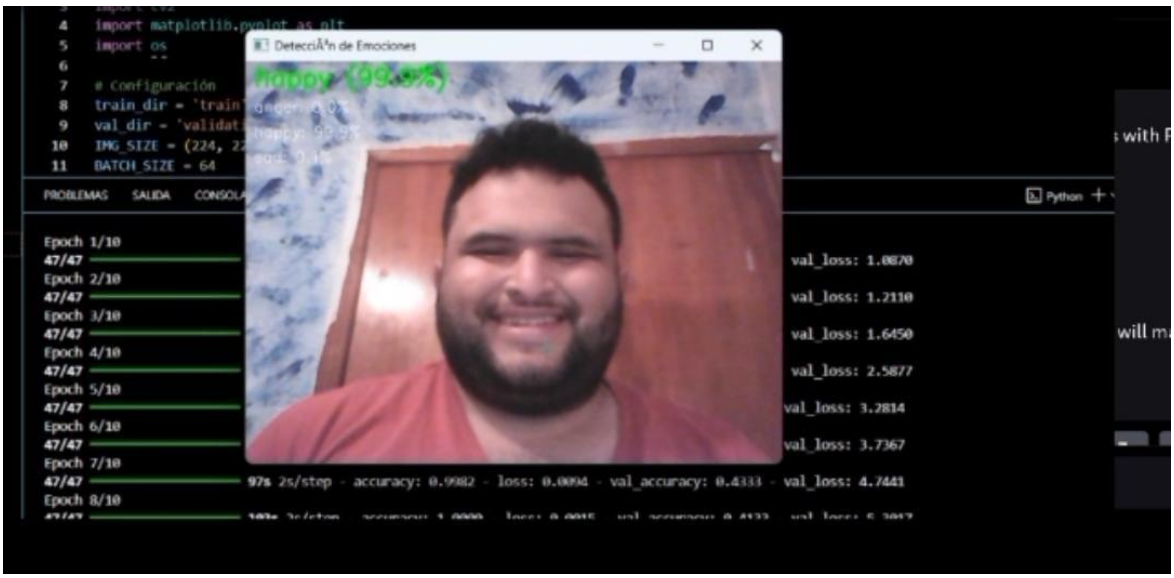
```
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(len(class_names), activation='softmax')
])
```

parámetros:

Parámetro	Valor Actual	Ubicación
Tamaño de imagen	(224, 224)	Parte superior del código
Tamaño de batch	64	Parte superior del código / en image_dataset_from_directory
Número de epoch	10	EPOCHS y en model.fit()

Las pruebas finales se realizaron en el video:

Capturas del video:

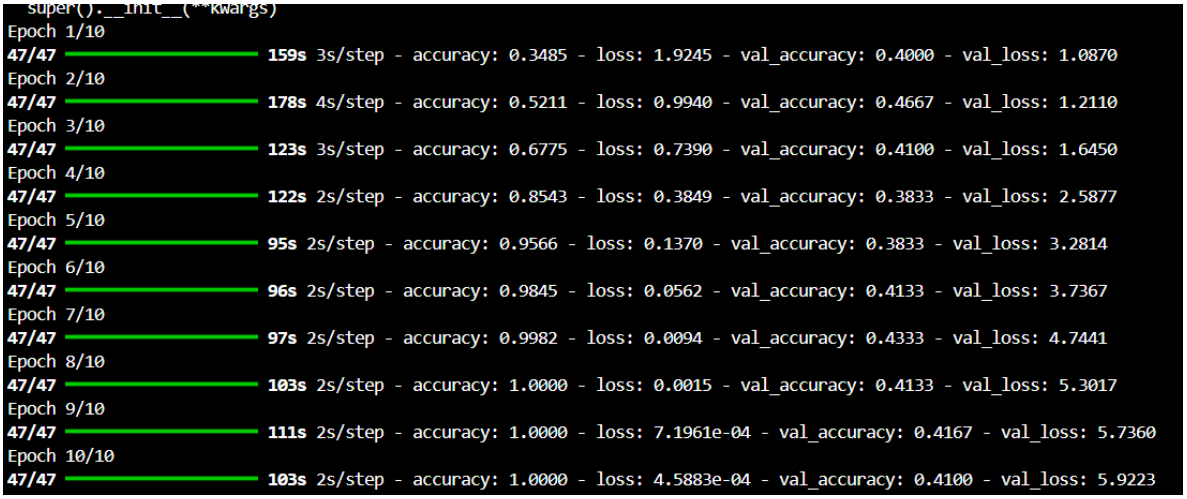


Modelo entrenándose:

Código: history = model.fit(train_ds, validation_data=val_ds, epochs=EPOCHS)

Explicación:

1. **Recorrido de los datos de entrenamiento (train_ds)**
    - El dataset train_ds contiene imágenes y sus etiquetas, agrupadas en lotes de 64 (batch_size=64).
    - Cada imagen se redimensiona a 224x224 píxeles y se normaliza al rango **[0, 1]** mediante la capa Rescaling, es una capa de preprocesamiento que sirve para normalizar los valores de los píxeles de las imágenes antes de que entren a la red neuronal.



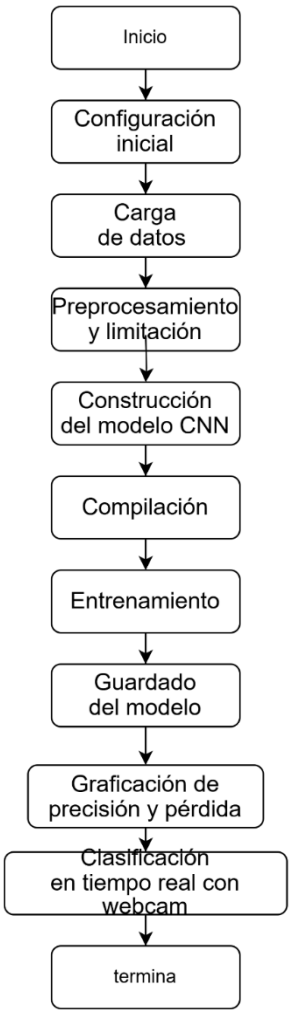
2. **Predicción:**
    - Por cada lote, el modelo procesa las imágenes y genera predicciones.
    - La última capa (Dense con softmax) devuelve probabilidades para cada emoción.
3. **Cálculo del error:**
    - Se compara la predicción del modelo con la etiqueta real usando categorical_crossentropy para medir qué tan equivocadas están las

prediccio

```
- accuracy: 0.3485 - loss: 1.9245 - val_accuracy: 0.4000 - val_loss: 1.0870

- accuracy: 0.5211 - loss: 0.9940 - val_accuracy: 0.4667 - val_loss: 1.2110

- accuracy: 0.6775 - loss: 0.7390 - val_accuracy: 0.4100 - val_loss: 1.6450

- accuracy: 0.8543 - loss: 0.3849 - val_accuracy: 0.3833 - val_loss: 2.5877

 accuracy: 0.9566 - loss: 0.1370 - val_accuracy: 0.3833 - val_loss: 3.2814

 accuracy: 0.9845 - loss: 0.0562 - val_accuracy: 0.4133 - val_loss: 3.7367

 accuracy: 0.9982 - loss: 0.0094 - val_accuracy: 0.4333 - val_loss: 4.7441

- accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 0.4133 - val_loss: 5.3017

- accuracy: 1.0000 - loss: 7.1961e-04 - val_accuracy: 0.4167 - val_loss: 5.7360
```

4. **Ajuste de los pesos:**
    - El optimizador adam calcula cómo ajustar los pesos de la red para reducir el error en el siguiente intento.
5. **Prueba con datos de validación:**
    - Al final de cada época, el modelo se evalúa con datos que no ha visto durante el entrenamiento.
6. **Registro del progreso**
    - El objeto history guarda métricas como:
        - **Precisión (accuracy)**: Porcentaje de aciertos en entrenamiento.
        - **Pérdida (loss)**: Error en entrenamiento.
        - **Val_accuracy & Val_loss**: Rendimiento con datos de validación.

Diagrama de flujo del programa:



Pasos:

1. **Configuración inicial.**
   IMG_SIZE = (224, 224)
   BATCH_SIZE = 64
   EPOCHS = 1
   MAX_IMAGES = 1000
2. **Carga de datos (train y validation).**
   train_ds = tf.keras.preprocessing.image_dataset_from_directory(...)
   val_ds = tf.keras.preprocessing.image_dataset_from_directory(...)
3. **Preprocesamiento y limitación.**
   class_names = train_ds.class_names
4. **Construcción del modelo CNN.**
   model = tf.keras.Sequential([
   tf.keras.layers.Rescaling(1./255, input_shape=(224, 224, 3)),
   tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
   tf.keras.layers.MaxPooling2D(),
   tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),

```
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(len(class_names), activation='softmax')
])
```

5. **Compilación.**

```
model.compile(optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy'])
```

6. **Entrenamiento.**

```
history = model.fit(train_ds, validation_data=val_ds, epochs=EPOCHS)
```

7. **Guardado del modelo.**

```
model.save('modelo_emociones.h5')
```

8. **Se grafica la precisión y pérdida.**

```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión')
plt.legend()
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida')
plt.legend()
plt.tight_layout()
plt.show()
```

Capturas:

9. **Clasificación en tiempo real con webcam.**

```
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print('No se pudo abrir la cámara')
    exit()

print("\nPresiona 'q' para salir")
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Preprocesamiento de la imagen en tiempo real
```

```
img = cv2.resize(frame, IMG_SIZE)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Asegura formato correcto
img_array = tf.expand_dims(img, 0)  # Añade dimensión batch

# Predicción
predictions = model.predict(img_array, verbose=0)
predicted_index = np.argmax(predictions[0])
emotion = class_names[predicted_index]
confidence = predictions[0][predicted_index] * 100

# Mostrar emoción principal
text = f'{emotion} ({confidence:.1f}%)'
cv2.putText(frame, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
        1, (0, 255, 0), 2, cv2.LINE_AA)
```

Bibliografía:

¿Qué son las redes neuronales convolucionales? | IBM

Enlace Video prueba:

https://youtu.be/5VQJnyktnNM?si=ObpQTCEGELVxRJ-r