Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

DIVISIÓN DE TECNOLOGÍAS PARA LA INTEGRACIÓN CIBER-HUMANA

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

Practica 2

TEMA: Redes Neuronales Convolucionales (CNN)

NOMBRE DEL ESTUDIANTE:

Padilla Perez Jorge Daray

NOMBRE DE LA MATERIA: Seminario de Solución de Problemas de Inteligencia Artificial II

SECCIÓN: D05

CALENDARIO: 2024-B

NOMBRE DEL PROFESOR: JAVIER AUGUSTO GALVIS CHACON

Contents

Resumen (CNN, temas vistos en clase)

Tema Principal: El documento es una presentación académica sobre Redes Neuronales Convolucionales (CNN) de la Universidad de Guadalajara, enfocada en el aprendizaje automático y el procesamiento de imágenes.

Definición y Propósito:

- Las CNN son redes neuronales artificiales que simulan el funcionamiento del córtex visual humano
- Su principal función es procesar e interpretar imágenes mediante aprendizaje supervisado
- Utilizan capas especializadas para identificar características desde básicas hasta complejas

Estructura y Funcionamiento:

- Procesamiento de Entrada:
    - Trabaja con imágenes (ejemplo: 28x28 píxeles)
    - Normaliza valores de píxeles (0-255) a valores entre 0 y 1

Componentes Principales: a) Convoluciones:

- Utiliza kernels para procesar grupos de píxeles
- Genera mapas de características
- Aplica múltiples filtros para detectar diferentes patrones

Subsampling:

- Reduce la cantidad de neuronas mediante Max-Pooling
- Mantiene las características más importantes
- Ayuda a optimizar el procesamiento computacional

Proceso de Aprendizaje:

- Utiliza backpropagation para ajustar los pesos de los kernels
- Requiere menos parámetros que una red neuronal tradicional
- Proceso jerárquico: desde características simples hasta complejas

Ventajas sobre Redes Neuronales Tradicionales:

- Menor cantidad de conexiones necesarias
- Mejor eficiencia en el procesamiento de imágenes
- Capacidad de detectar características jerárquicas
- Mayor especialización en el reconocimiento de patrones visuales

Enunciado del problema

El aumento de la digitalización de los eventos deportivos ha generado una gran cantidad de imágenes y videos en tiempo real que requieren ser clasificados y procesados automáticamente. Un escenario real es el de las transmisiones de eventos multideportivos, como los Juegos Olímpicos, donde los algoritmos deben identificar correctamente las disciplinas deportivas en imágenes capturadas de diferentes ángulos y en condiciones variables de iluminación y movimiento. La clasificación automática de imágenes deportivas puede facilitar tareas como la generación automática de estadísticas, resúmenes visuales, e incluso mejorar la experiencia del espectador con sugerencias personalizadas de contenido.

Codigo utilizado

CNN desde cero

```
import tensorflow as tf
from keras._tf_keras.keras.preprocessing.image import ImageDataGenerator
from keras._tf_keras.keras.models import Sequential
from keras._tf_keras.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout, BatchNormalization
from keras._tf_keras.keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# Directorio de entrenamiento
train_dir = 'dataset'

# generador de datos
train_datagen = ImageDataGenerator(
    rescale=1./255,          # Escalar los valores de los píxeles a [0, 1]
    shear_range=0.2,         # Aplicar transformaciones de corte
    zoom_range=0.2,          # Aplicar zoom a las imágenes
    horizontal_flip=True,     # Voltear horizontalmente
    validation_split=0.2)     # Separar el 20% de los datos para validación

# datos de entrenamiento
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(64, 64),    # Cambiar tamaño de imágenes a 64x64 píxeles
```

```
    batch_size=32,
    class_mode='categorical',
    subset='training')       # Usar los datos para entrenamiento

# datos de validación
validation_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    subset='validation')     # Usar los datos para validación

# modelo secuencial
model = Sequential()

# Capa convolucional 1
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Capa convolucional 2
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Capa convolucional 3
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Aplanar las capas convolucionales
model.add(Flatten())

# Capa densa 1
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

# Capa de salida
model.add(Dense(10, activation='softmax'))  # Cambiar por el número de
clases deportivas caso deportes 10

# Adam ajustado y categorical_crossentropy
```

```python
model.compile(optimizer=Adam(learning_rate=0.0001),  # Taza de aprendizaje
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Entrenar el modelo
history = model.fit(
    train_generator,
    epochs=20,
    validation_data=validation_generator
)

# Resumen del modelo
model.summary()

# Guardar el modelo entrenado para programa predecir
model.save('intento5_deportes.h5')

# Reiniciar el generador de validación
validation_generator.reset()

# predicciones del modelo en el conjunto de validación
Y_true = validation_generator.classes
Y_pred = model.predict(validation_generator)
Y_pred_classes = np.argmax(Y_pred, axis=1)

# Etiquetas de las clases
class_labels = list(validation_generator.class_indices.keys())

# Reporte de clasificación
print("Reporte de clasificación para el conjunto de validación:\n")
print(classification_report(Y_true, Y_pred_classes,
target_names=class_labels))

# Matriz de confusión
conf_matrix = confusion_matrix(Y_true, Y_pred_classes)
print("Matriz de Confusión:\n", conf_matrix)

# Evaluación de precisión y pérdida en entrenamiento y validación
train_loss, train_accuracy = model.evaluate(train_generator, verbose=0)
val_loss, val_accuracy = model.evaluate(validation_generator, verbose=0)
print(f"\nPérdida en entrenamiento: {train_loss:.4f}, Precisión en
entrenamiento: {train_accuracy:.4f}")
print(f"Pérdida en validación: {val_loss:.4f}, Precisión en validación:
{val_accuracy:.4f}")
```

Transfer Learning

```
import tensorflow as tf
from keras._tf_keras.keras.applications import VGG16
from keras._tf_keras.keras.layers import Dense, Flatten, Dropout
from keras._tf_keras.keras.models import Model
from keras._tf_keras.keras.optimizers import Adam
from keras._tf_keras.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras._tf_keras.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# Directorio de entrenamiento
train_dir = 'dataset'

# datos con aumento de datos para el entrenamiento
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)

# datos de entrenamiento
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical',
    subset='training')

# datos de validación
validation_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical',
    subset='validation')

# Cargar el modelo preentrenado VGG16
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(128,
128, 3))

# capas convolucionales
```

```
for layer in base_model.layers:
    layer.trainable = False

# nuevas capas densas al final del modelo
x = base_model.output
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(10, activation='softmax')(x)

# Crear el modelo final
model = Model(inputs=base_model.input, outputs=output)

# Adam y categorical_crossentropy
model.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Entrenamiento inicial
history_initial = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    callbacks=[EarlyStopping(monitor='val_loss', patience=3)]
)

# capas del modelo base para fine-tuning
for layer in base_model.layers[-10:]:
    layer.trainable = True

# Compilar nuevamente el modelo para fine-tuning
model.compile(
    optimizer=Adam(learning_rate=0.00001), # Taza de entrenamiento
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# fine-tuning
history_fine_tuning = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
```

```
    callbacks=[ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=3)]
)

# Guardar el modelo entrenado
model.save('modelo2_fine_tuning_deportes.h5')

# Evaluación del modelo inicial
print("\nEvaluación del modelo inicial:")
validation_generator.reset()
Y_true = validation_generator.classes
Y_pred_initial = model.predict(validation_generator, verbose=1)
Y_pred_classes_initial = np.argmax(Y_pred_initial, axis=1)

# Reporte de clasificación para el modelo inicial
class_labels = list(validation_generator.class_indices.keys())
print("Reporte de clasificación para el modelo inicial:\n")
print(classification_report(Y_true, Y_pred_classes_initial,
target_names=class_labels))

# Matriz de confusión para el modelo inicial
conf_matrix_initial = confusion_matrix(Y_true, Y_pred_classes_initial)
print("Matriz de Confusión para el modelo inicial:\n", conf_matrix_initial)

# Evaluación de precisión y pérdida en entrenamiento y validación del modelo
inicial
train_loss_initial, train_accuracy_initial = model.evaluate(train_generator,
verbose=0)
val_loss_initial, val_accuracy_initial =
model.evaluate(validation_generator, verbose=0)
print(f"\nPérdida en entrenamiento (inicial): {train_loss_initial:.4f},
Precisión en entrenamiento: {train_accuracy_initial:.4f}")
print(f"Pérdida en validación (inicial): {val_loss_initial:.4f}, Precisión
en validación: {val_accuracy_initial:.4f}")

# Evaluación del modelo con fine-tuning con métricas adicionales
print("\nEvaluación del modelo con fine-tuning:")
validation_generator.reset()
Y_pred_fine_tuning = model.predict(validation_generator, verbose=1)
Y_pred_classes_fine_tuning = np.argmax(Y_pred_fine_tuning, axis=1)

# Reporte de clasificación para el modelo con fine-tuning
print("Reporte de clasificación para el modelo con fine-tuning:\n")
print(classification_report(Y_true, Y_pred_classes_fine_tuning,
target_names=class_labels))
```

```python
# Matriz de confusión para el modelo con fine-tuning
conf_matrix_fine_tuning = confusion_matrix(Y_true,
Y_pred_classes_fine_tuning)
print("Matriz de Confusión para el modelo con fine-tuning:\n",
conf_matrix_fine_tuning)

# Evaluación de precisión y pérdida en entrenamiento y validación del modelo
con fine-tuning
train_loss_fine, train_accuracy_fine = model.evaluate(train_generator,
verbose=0)
val_loss_fine, val_accuracy_fine = model.evaluate(validation_generator,
verbose=0)
print(f"\nPérdida en entrenamiento (fine-tuning): {train_loss_fine:.4f},
Precisión en entrenamiento: {train_accuracy_fine:.4f}")
print(f"Pérdida en validación (fine-tuning): {val_loss_fine:.4f}, Precisión
en validación: {val_accuracy_fine:.4f}")
```

Predecir

```python
from keras._tf_keras.keras.models import load_model
from keras._tf_keras.keras.utils import load_img, img_to_array
import numpy as np
# Cargar el modelo entrenado
model = load_model('intento1_deportes.h5')

# Cargar y preprocesar la imagen para que coincida con el tamaño esperado
por el modelo
test_image = load_img('single_test/tenis.jpg', target_size=(64, 64))  #
Cambia la ruta a tu imagen y ajusta el tamaño al que entrenaste (64x64)
test_image = img_to_array(test_image)

# Normalizar la imagen como en el entrenamiento
test_image = test_image / 255.0

# Expandir las dimensiones para hacer la predicción (1, 64, 64, 3)
test_image = np.expand_dims(test_image, axis=0)

# Hacer la predicción
result = model.predict(test_image)

# Mostrar los valores predichos (probabilidades para cada clase)
print("Probabilidades predichas:", result)
```

```
# Obtener el índice de la clase con mayor probabilidad
predicted_class_index = np.argmax(result)

# Obtener el mapeo de clases (esto necesita el generador de datos de entrenamiento)
# Puedes guardar el mapeo de clases al momento de entrenamiento o cargarlo desde training_dataset.class_indices si está disponible
class_labels = {0: 'ajedrez', 1: 'baloncesto', 2: 'boxeo', 3: 'disparo', 4: 'esgrima',
                5: 'formula1', 6: 'futbol', 7: 'hockey', 8: 'natacion', 9: 'tenis'}  # Ejemplo de cómo podrías mapear las clases

# Obtener el nombre de la clase predicha
predicted_class_label = class_labels[predicted_class_index]

# Imprimir la clase predicha
print(f'La imagen pertenece a la clase: {predicted_class_label}')
```

Resultados

a) Prueba 1: Modelo CNN desde cero

Epoch 1/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 152s 883ms/step - accuracy: 0.2487 - loss: 2.6821 - val_accuracy: 0.1998 - val_loss: 5.1386

Epoch 2/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.3395 - loss: 1.9211 - val_accuracy: 0.2245 - val_loss: 3.0953

Epoch 3/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.3996 - loss: 1.7852 - val_accuracy: 0.4212 - val_loss: 1.8967

Epoch 4/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.4148 - loss: 1.7371 - val_accuracy: 0.5215 - val_loss: 1.5496

Epoch 5/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.4279 - loss: 1.7086 - val_accuracy: 0.5478 - val_loss: 1.4147

Epoch 6/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.4574 - loss: 1.5929 - val_accuracy: 0.5669 - val_loss: 1.2986

Epoch 7/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.4968 - loss: 1.4962 - val_accuracy: 0.5868 - val_loss: 1.2330

Epoch 8/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.4766 - loss: 1.5104 - val_accuracy: 0.6330 - val_loss: 1.0780

Epoch 9/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.5023 - loss: 1.4676 - val_accuracy: 0.6513 - val_loss: 1.0867

Epoch 10/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.5299 - loss: 1.4184 - val_accuracy: 0.5725 - val_loss: 1.2591

Epoch 11/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.5287 - loss: 1.4071 - val_accuracy: 0.6003 - val_loss: 1.2434

Epoch 12/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.5321 - loss: 1.3831 - val_accuracy: 0.6218 - val_loss: 1.1531

Epoch 13/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.5538 - loss: 1.3188 - val_accuracy: 0.6831 - val_loss: 0.9791

Epoch 14/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.5608 - loss: 1.3217 - val_accuracy: 0.7134 - val_loss: 0.8552

Epoch 15/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.5862 - loss: 1.2284 - val_accuracy: 0.6632 - val_loss: 1.1249

Epoch 16/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.5775 - loss: 1.2596 - val_accuracy: 0.7412 - val_loss: 0.8851

Epoch 17/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.5958 - loss: 1.2370 - val_accuracy: 0.7484 - val_loss: 0.7892

Epoch 18/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.6074 - loss: 1.1697 - val_accuracy: 0.7436 - val_loss: 0.8292

Epoch 19/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.6020 - loss: 1.1881 - val_accuracy: 0.6887 - val_loss: 0.9887

Epoch 20/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.6175 - loss: 1.1792 - val_accuracy: 0.7142 - val_loss: 0.9443

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 32)	896
batch_normalization (BatchNormalization)	(None, 62, 62, 32)	128
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0
dropout (Dropout)	(None, 31, 31, 32)	0
conv2d_1 (Conv2D)	(None, 29, 29, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 29, 29, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_1 (Dropout)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 12, 12, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 12, 12, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 128)	0
dropout_2 (Dropout)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 128)	589,952
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 2,055,264 (7.84 MB)

Trainable params: 684,938 (2.61 MB)

Non-trainable params: 448 (1.75 KB)

Optimizer params: 1,369,878 (5.23 MB)

b) Prueba 2: Modelo con Transfer Learning

Vgg16

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5

58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 14s 0us/step

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 128, 128, 3)	0
block1_conv1 (Conv2D)	(None, 128, 128, 64)	1,792
block1_conv2 (Conv2D)	(None, 128, 128, 64)	36,928
block1_pool (MaxPooling2D)	(None, 64, 64, 64)	0
block2_conv1 (Conv2D)	(None, 64, 64, 128)	73,856
block2_conv2 (Conv2D)	(None, 64, 64, 128)	147,584
block2_pool (MaxPooling2D)	(None, 32, 32, 128)	0
block3_conv1 (Conv2D)	(None, 32, 32, 256)	295,168
block3_conv2 (Conv2D)	(None, 32, 32, 256)	590,080
block3_conv3 (Conv2D)	(None, 32, 32, 256)	590,080
block3_pool (MaxPooling2D)	(None, 16, 16, 256)	0
block4_conv1 (Conv2D)	(None, 16, 16, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 16, 16, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 16, 16, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 8, 8, 512)	0
block5_conv1 (Conv2D)	(None, 8, 8, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 8, 8, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 8, 8, 512)	2,359,808

block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 128)	1,048,704
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Model: "functional"

Total params: 15,764,682 (60.14 MB)

Trainable params: 1,049,994 (4.01 MB)

Non-trainable params: 14,714,688 (56.13 MB)

ResNet50

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5

94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 8s 0us/step

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 128, 128, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 134, 134, 3)	0	input_layer[0][0]
conv1_conv (Conv2D)	(None, 64, 64, 64)	9,472	conv1_pad[0][0]
conv1_bn (BatchNormalization)	(None, 64, 64, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 64, 64, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 66, 66, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 32, 32, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 32, 32, 64)	4,160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalization)	(None, 32, 32, 64)	256	conv2_block1_1_conv[0][0]
conv2_block1_1_relu (Activation)	(None, 32, 32, 64)	0	conv2_block1_1_bn[0][0]
conv2_block1_2_conv (Conv2D)	(None, 32, 32, 64)	36,928	conv2_block1_1_relu[0][0]
conv2_block1_2_bn (BatchNormalization)	(None, 32, 32, 64)	256	conv2_block1_2_conv[0][0]
conv2_block1_2_relu (Activation)	(None, 32, 32, 64)	0	conv2_block1_2_bn[0][0]
conv2_block1_0_conv (Conv2D)	(None, 32, 32, 256)	16,640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 32, 32, 256)	16,640	conv2_block1_2_relu[0][0]
conv2_block1_0_bn (BatchNormalization)	(None, 32, 32, 256)	1,024	conv2_block1_0_conv[0][0]
conv2_block1_3_bn (BatchNormalization)	(None, 32, 32, 256)	1,024	conv2_block1_3_conv[0][0]

conv2_block1_add (Add)	(None, 32, 32, 256)	0	conv2_block1_0_bn[0][0], conv2_block1_3_bn[0][0]
conv2_block1_out (Activation)	(None, 32, 32, 256)	0	conv2_block1_add[0][0]
conv2_block2_1_conv (Conv2D)	(None, 32, 32, 64)	16,448	conv2_block1_out[0][0]
conv2_block2_1_bn (BatchNormalization)	(None, 32, 32, 64)	256	conv2_block2_1_conv[0][0]
conv2_block2_1_relu (Activation)	(None, 32, 32, 64)	0	conv2_block2_1_bn[0][0]
conv2_block2_2_conv (Conv2D)	(None, 32, 32, 64)	36,928	conv2_block2_1_relu[0][0]
conv2_block2_2_bn (BatchNormalization)	(None, 32, 32, 64)	256	conv2_block2_2_conv[0][0]
conv2_block2_2_relu (Activation)	(None, 32, 32, 64)	0	conv2_block2_2_bn[0][0]
conv2_block2_3_conv (Conv2D)	(None, 32, 32, 256)	16,640	conv2_block2_2_relu[0][0]
conv2_block2_3_bn (BatchNormalization)	(None, 32, 32, 256)	1,024	conv2_block2_3_conv[0][0]
conv2_block2_add (Add)	(None, 32, 32, 256)	0	conv2_block1_out[0][0], conv2_block2_3_bn[0][0]
conv2_block2_out (Activation)	(None, 32, 32, 256)	0	conv2_block2_add[0][0]
conv2_block3_1_conv (Conv2D)	(None, 32, 32, 64)	16,448	conv2_block2_out[0][0]
conv2_block3_1_bn (BatchNormalization)	(None, 32, 32, 64)	256	conv2_block3_1_conv[0][0]
conv2_block3_1_relu (Activation)	(None, 32, 32, 64)	0	conv2_block3_1_bn[0][0]
conv2_block3_2_conv (Conv2D)	(None, 32, 32, 64)	36,928	conv2_block3_1_relu[0][0]

conv2_block3_2_bn (BatchNormalization)	(None, 32, 32, 64)	256	conv2_block3_2_conv[0][0]
conv2_block3_2_relu (Activation)	(None, 32, 32, 64)	0	conv2_block3_2_bn[0][0]
conv2_block3_3_conv (Conv2D)	(None, 32, 32, 256)	16,640	conv2_block3_2_relu[0][0]
conv2_block3_3_bn (BatchNormalization)	(None, 32, 32, 256)	1,024	conv2_block3_3_conv[0][0]
conv2_block3_add (Add)	(None, 32, 32, 256)	0	conv2_block2_out[0][0], conv2_block3_3_bn[0][0]
conv2_block3_out (Activation)	(None, 32, 32, 256)	0	conv2_block3_add[0][0]
conv3_block1_1_conv (Conv2D)	(None, 16, 16, 128)	32,896	conv2_block3_out[0][0]
conv3_block1_1_bn (BatchNormalization)	(None, 16, 16, 128)	512	conv3_block1_1_conv[0][0]
conv3_block1_1_relu (Activation)	(None, 16, 16, 128)	0	conv3_block1_1_bn[0][0]
conv3_block1_2_conv (Conv2D)	(None, 16, 16, 128)	147,584	conv3_block1_1_relu[0][0]
conv3_block1_2_bn (BatchNormalization)	(None, 16, 16, 128)	512	conv3_block1_2_conv[0][0]
conv3_block1_2_relu (Activation)	(None, 16, 16, 128)	0	conv3_block1_2_bn[0][0]
conv3_block1_0_conv (Conv2D)	(None, 16, 16, 512)	131,584	conv2_block3_out[0][0]
conv3_block1_3_conv (Conv2D)	(None, 16, 16, 512)	66,048	conv3_block1_2_relu[0][0]
conv3_block1_0_bn (BatchNormalization)	(None, 16, 16, 512)	2,048	conv3_block1_0_conv[0][0]

conv3_block1_3_bn (BatchNormalization)	(None, 16, 16, 512)	2,048	conv3_block1_3_conv[0][0]
conv3_block1_add (Add)	(None, 16, 16, 512)	0	conv3_block1_0_bn[0][0], conv3_block1_3_bn[0][0]
conv3_block1_out (Activation)	(None, 16, 16, 512)	0	conv3_block1_add[0][0]
conv3_block2_1_conv (Conv2D)	(None, 16, 16, 128)	65,664	conv3_block1_out[0][0]
conv3_block2_1_bn (BatchNormalization)	(None, 16, 16, 128)	512	conv3_block2_1_conv[0][0]
conv3_block2_1_relu (Activation)	(None, 16, 16, 128)	0	conv3_block2_1_bn[0][0]
conv3_block2_2_conv (Conv2D)	(None, 16, 16, 128)	147,584	conv3_block2_1_relu[0][0]
conv3_block2_2_bn (BatchNormalization)	(None, 16, 16, 128)	512	conv3_block2_2_conv[0][0]
conv3_block2_2_relu (Activation)	(None, 16, 16, 128)	0	conv3_block2_2_bn[0][0]
conv3_block2_3_conv (Conv2D)	(None, 16, 16, 512)	66,048	conv3_block2_2_relu[0][0]
conv3_block2_3_bn (BatchNormalization)	(None, 16, 16, 512)	2,048	conv3_block2_3_conv[0][0]
conv3_block2_add (Add)	(None, 16, 16, 512)	0	conv3_block1_out[0][0], conv3_block2_3_bn[0][0]
conv3_block2_out (Activation)	(None, 16, 16, 512)	0	conv3_block2_add[0][0]
conv3_block3_1_conv (Conv2D)	(None, 16, 16, 128)	65,664	conv3_block2_out[0][0]
conv3_block3_1_bn (BatchNormalization)	(None, 16, 16, 128)	512	conv3_block3_1_conv[0][0]

conv3_block3_1_relu (Activation)	(None, 16, 16, 128)	0	conv3_block3_1_bn[0][0]
conv3_block3_2_conv (Conv2D)	(None, 16, 16, 128)	147,584	conv3_block3_1_relu[0][0]
conv3_block3_2_bn (BatchNormalization)	(None, 16, 16, 128)	512	conv3_block3_2_conv[0][0]
conv3_block3_2_relu (Activation)	(None, 16, 16, 128)	0	conv3_block3_2_bn[0][0]
conv3_block3_3_conv (Conv2D)	(None, 16, 16, 512)	66,048	conv3_block3_2_relu[0][0]
conv3_block3_3_bn (BatchNormalization)	(None, 16, 16, 512)	2,048	conv3_block3_3_conv[0][0]
conv3_block3_add (Add)	(None, 16, 16, 512)	0	conv3_block2_out[0][0], conv3_block3_3_bn[0][0]
conv3_block3_out (Activation)	(None, 16, 16, 512)	0	conv3_block3_add[0][0]
conv3_block4_1_conv (Conv2D)	(None, 16, 16, 128)	65,664	conv3_block3_out[0][0]
conv3_block4_1_bn (BatchNormalization)	(None, 16, 16, 128)	512	conv3_block4_1_conv[0][0]
conv3_block4_1_relu (Activation)	(None, 16, 16, 128)	0	conv3_block4_1_bn[0][0]
conv3_block4_2_conv (Conv2D)	(None, 16, 16, 128)	147,584	conv3_block4_1_relu[0][0]
conv3_block4_2_bn (BatchNormalization)	(None, 16, 16, 128)	512	conv3_block4_2_conv[0][0]
conv3_block4_2_relu (Activation)	(None, 16, 16, 128)	0	conv3_block4_2_bn[0][0]
conv3_block4_3_conv (Conv2D)	(None, 16, 16, 512)	66,048	conv3_block4_2_relu[0][0]
conv3_block4_3_bn (BatchNormalization)	(None, 16, 16, 512)	2,048	conv3_block4_3_conv[0][0]

conv3_block4_add (Add)	(None, 16, 16, 512)	0	conv3_block3_out[0][0], conv3_block4_3_bn[0][0]
conv3_block4_out (Activation)	(None, 16, 16, 512)	0	conv3_block4_add[0][0]
conv4_block1_1_conv (Conv2D)	(None, 8, 8, 256)	131,328	conv3_block4_out[0][0]
conv4_block1_1_bn (BatchNormalization)	(None, 8, 8, 256)	1,024	conv4_block1_1_conv[0][0]
conv4_block1_1_relu (Activation)	(None, 8, 8, 256)	0	conv4_block1_1_bn[0][0]
conv4_block1_2_conv (Conv2D)	(None, 8, 8, 256)	590,080	conv4_block1_1_relu[0][0]
conv4_block1_2_bn (BatchNormalization)	(None, 8, 8, 256)	1,024	conv4_block1_2_conv[0][0]
conv4_block1_2_relu (Activation)	(None, 8, 8, 256)	0	conv4_block1_2_bn[0][0]
conv4_block1_0_conv (Conv2D)	(None, 8, 8, 1024)	525,312	conv3_block4_out[0][0]
conv4_block1_3_conv (Conv2D)	(None, 8, 8, 1024)	263,168	conv4_block1_2_relu[0][0]
conv4_block1_0_bn (BatchNormalization)	(None, 8, 8, 1024)	4,096	conv4_block1_0_conv[0][0]
conv4_block1_3_bn (BatchNormalization)	(None, 8, 8, 1024)	4,096	conv4_block1_3_conv[0][0]
conv4_block1_add (Add)	(None, 8, 8, 1024)	0	conv4_block1_0_bn[0][0], conv4_block1_3_bn[0][0]
conv4_block1_out (Activation)	(None, 8, 8, 1024)	0	conv4_block1_add[0][0]
conv4_block2_1_conv (Conv2D)	(None, 8, 8, 256)	262,400	conv4_block1_out[0][0]
conv4_block2_1_bn (BatchNormalization)	(None, 8, 8, 256)	1,024	conv4_block2_1_conv[0][0]

conv4_block2_1_relu (Activation)	(None, 8, 8, 256)	0	conv4_block2_1_bn[0][0]
conv4_block2_2_conv (Conv2D)	(None, 8, 8, 256)	590,080	conv4_block2_1_relu[0][0]
conv4_block2_2_bn (BatchNormalization)	(None, 8, 8, 256)	1,024	conv4_block2_2_conv[0][0]
conv4_block2_2_relu (Activation)	(None, 8, 8, 256)	0	conv4_block2_2_bn[0][0]
conv4_block2_3_conv (Conv2D)	(None, 8, 8, 1024)	263,168	conv4_block2_2_relu[0][0]
conv4_block2_3_bn (BatchNormalization)	(None, 8, 8, 1024)	4,096	conv4_block2_3_conv[0][0]
conv4_block2_add (Add)	(None, 8, 8, 1024)	0	conv4_block1_out[0][0], conv4_block2_3_bn[0][0]
conv4_block2_out (Activation)	(None, 8, 8, 1024)	0	conv4_block2_add[0][0]
conv4_block3_1_conv (Conv2D)	(None, 8, 8, 256)	262,400	conv4_block2_out[0][0]
conv4_block3_1_bn (BatchNormalization)	(None, 8, 8, 256)	1,024	conv4_block3_1_conv[0][0]
conv4_block3_1_relu (Activation)	(None, 8, 8, 256)	0	conv4_block3_1_bn[0][0]
conv4_block3_2_conv (Conv2D)	(None, 8, 8, 256)	590,080	conv4_block3_1_relu[0][0]
conv4_block3_2_bn (BatchNormalization)	(None, 8, 8, 256)	1,024	conv4_block3_2_conv[0][0]
conv4_block3_2_relu (Activation)	(None, 8, 8, 256)	0	conv4_block3_2_bn[0][0]
conv4_block3_3_conv (Conv2D)	(None, 8, 8, 1024)	263,168	conv4_block3_2_relu[0][0]

conv4_block3_3_bn (BatchNormalization)	(None, 8, 8, 1024)	4,096	conv4_block3_3_conv[0][0]
conv4_block3_add (Add)	(None, 8, 8, 1024)	0	conv4_block2_out[0][0], conv4_block3_3_bn[0][0]
conv4_block3_out (Activation)	(None, 8, 8, 1024)	0	conv4_block3_add[0][0]
conv4_block4_1_conv (Conv2D)	(None, 8, 8, 256)	262,400	conv4_block3_out[0][0]
conv4_block4_1_bn (BatchNormalization)	(None, 8, 8, 256)	1,024	conv4_block4_1_conv[0][0]
conv4_block4_1_relu (Activation)	(None, 8, 8, 256)	0	conv4_block4_1_bn[0][0]
conv4_block4_2_conv (Conv2D)	(None, 8, 8, 256)	590,080	conv4_block4_1_relu[0][0]
conv4_block4_2_bn (BatchNormalization)	(None, 8, 8, 256)	1,024	conv4_block4_2_conv[0][0]
conv4_block4_2_relu (Activation)	(None, 8, 8, 256)	0	conv4_block4_2_bn[0][0]
conv4_block4_3_conv (Conv2D)	(None, 8, 8, 1024)	263,168	conv4_block4_2_relu[0][0]
conv4_block4_3_bn (BatchNormalization)	(None, 8, 8, 1024)	4,096	conv4_block4_3_conv[0][0]
conv4_block4_add (Add)	(None, 8, 8, 1024)	0	conv4_block3_out[0][0], conv4_block4_3_bn[0][0]
conv4_block4_out (Activation)	(None, 8, 8, 1024)	0	conv4_block4_add[0][0]
conv4_block5_1_conv (Conv2D)	(None, 8, 8, 256)	262,400	conv4_block4_out[0][0]
conv4_block5_1_bn (BatchNormalization)	(None, 8, 8, 256)	1,024	conv4_block5_1_conv[0][0]
conv4_block5_1_relu (Activation)	(None, 8, 8, 256)	0	conv4_block5_1_bn[0][0]

conv4_block5_2_conv (Conv2D)	(None, 8, 8, 256)	590,080	conv4_block5_1_relu[0][0]
conv4_block5_2_bn (BatchNormalization)	(None, 8, 8, 256)	1,024	conv4_block5_2_conv[0][0]
conv4_block5_2_relu (Activation)	(None, 8, 8, 256)	0	conv4_block5_2_bn[0][0]
conv4_block5_3_conv (Conv2D)	(None, 8, 8, 1024)	263,168	conv4_block5_2_relu[0][0]
conv4_block5_3_bn (BatchNormalization)	(None, 8, 8, 1024)	4,096	conv4_block5_3_conv[0][0]
conv4_block5_add (Add)	(None, 8, 8, 1024)	0	conv4_block4_out[0][0], conv4_block5_3_bn[0][0]
conv4_block5_out (Activation)	(None, 8, 8, 1024)	0	conv4_block5_add[0][0]
conv4_block6_1_conv (Conv2D)	(None, 8, 8, 256)	262,400	conv4_block5_out[0][0]
conv4_block6_1_bn (BatchNormalization)	(None, 8, 8, 256)	1,024	conv4_block6_1_conv[0][0]
conv4_block6_1_relu (Activation)	(None, 8, 8, 256)	0	conv4_block6_1_bn[0][0]
conv4_block6_2_conv (Conv2D)	(None, 8, 8, 256)	590,080	conv4_block6_1_relu[0][0]
conv4_block6_2_bn (BatchNormalization)	(None, 8, 8, 256)	1,024	conv4_block6_2_conv[0][0]
conv4_block6_2_relu (Activation)	(None, 8, 8, 256)	0	conv4_block6_2_bn[0][0]
conv4_block6_3_conv (Conv2D)	(None, 8, 8, 1024)	263,168	conv4_block6_2_relu[0][0]
conv4_block6_3_bn (BatchNormalization)	(None, 8, 8, 1024)	4,096	conv4_block6_3_conv[0][0]
conv4_block6_add (Add)	(None, 8, 8, 1024)	0	conv4_block5_out[0][0], conv4_block6_3_bn[0][0]

conv4_block6_out (Activation)	(None, 8, 8, 1024)	0	conv4_block6_add[0][0]
conv5_block1_1_conv (Conv2D)	(None, 4, 4, 512)	524,800	conv4_block6_out[0][0]
conv5_block1_1_bn (BatchNormalization)	(None, 4, 4, 512)	2,048	conv5_block1_1_conv[0][0]
conv5_block1_1_relu (Activation)	(None, 4, 4, 512)	0	conv5_block1_1_bn[0][0]
conv5_block1_2_conv (Conv2D)	(None, 4, 4, 512)	2,359,808	conv5_block1_1_relu[0][0]
conv5_block1_2_bn (BatchNormalization)	(None, 4, 4, 512)	2,048	conv5_block1_2_conv[0][0]
conv5_block1_2_relu (Activation)	(None, 4, 4, 512)	0	conv5_block1_2_bn[0][0]
conv5_block1_0_conv (Conv2D)	(None, 4, 4, 2048)	2,099,200	conv4_block6_out[0][0]
conv5_block1_3_conv (Conv2D)	(None, 4, 4, 2048)	1,050,624	conv5_block1_2_relu[0][0]
conv5_block1_0_bn (BatchNormalization)	(None, 4, 4, 2048)	8,192	conv5_block1_0_conv[0][0]
conv5_block1_3_bn (BatchNormalization)	(None, 4, 4, 2048)	8,192	conv5_block1_3_conv[0][0]
conv5_block1_add (Add)	(None, 4, 4, 2048)	0	conv5_block1_0_bn[0][0], conv5_block1_3_bn[0][0]
conv5_block1_out (Activation)	(None, 4, 4, 2048)	0	conv5_block1_add[0][0]
conv5_block2_1_conv (Conv2D)	(None, 4, 4, 512)	1,049,088	conv5_block1_out[0][0]
conv5_block2_1_bn (BatchNormalization)	(None, 4, 4, 512)	2,048	conv5_block2_1_conv[0][0]
conv5_block2_1_relu (Activation)	(None, 4, 4, 512)	0	conv5_block2_1_bn[0][0]
conv5_block2_2_conv (Conv2D)	(None, 4, 4, 512)	2,359,808	conv5_block2_1_relu[0][0]

conv5_block2_2_bn (BatchNormalization)	(None, 4, 4, 512)	2,048	conv5_block2_2_conv[0][0]
conv5_block2_2_relu (Activation)	(None, 4, 4, 512)	0	conv5_block2_2_bn[0][0]
conv5_block2_3_conv (Conv2D)	(None, 4, 4, 2048)	1,050,624	conv5_block2_2_relu[0][0]
conv5_block2_3_bn (BatchNormalization)	(None, 4, 4, 2048)	8,192	conv5_block2_3_conv[0][0]
conv5_block2_add (Add)	(None, 4, 4, 2048)	0	conv5_block1_out[0][0], conv5_block2_3_bn[0][0]
conv5_block2_out (Activation)	(None, 4, 4, 2048)	0	conv5_block2_add[0][0]
conv5_block3_1_conv (Conv2D)	(None, 4, 4, 512)	1,049,088	conv5_block2_out[0][0]
conv5_block3_1_bn (BatchNormalization)	(None, 4, 4, 512)	2,048	conv5_block3_1_conv[0][0]
conv5_block3_1_relu (Activation)	(None, 4, 4, 512)	0	conv5_block3_1_bn[0][0]
conv5_block3_2_conv (Conv2D)	(None, 4, 4, 512)	2,359,808	conv5_block3_1_relu[0][0]
conv5_block3_2_bn (BatchNormalization)	(None, 4, 4, 512)	2,048	conv5_block3_2_conv[0][0]
conv5_block3_2_relu (Activation)	(None, 4, 4, 512)	0	conv5_block3_2_bn[0][0]
conv5_block3_3_conv (Conv2D)	(None, 4, 4, 2048)	1,050,624	conv5_block3_2_relu[0][0]
conv5_block3_3_bn (BatchNormalization)	(None, 4, 4, 2048)	8,192	conv5_block3_3_conv[0][0]
conv5_block3_add (Add)	(None, 4, 4, 2048)	0	conv5_block2_out[0][0], conv5_block3_3_bn[0][0]
conv5_block3_out (Activation)	(None, 4, 4, 2048)	0	conv5_block3_add[0][0]
flatten (Flatten)	(None, 32768)	0	conv5_block3_out[0][0]
dense (Dense)	(None, 128)	4,194,432	flatten[0][0]
dropout (Dropout)	(None, 128)	0	dense[0][0]
dense_1 (Dense)	(None, 10)	1,290	dropout[0][0]

Total params: 27,783,434 (105.99 MB)

Trainable params: 4,195,722 (16.01 MB)

Non-trainable params: 23,587,712 (89.98 MB)

InceptionV3

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5

87910968/87910968 ━━━━━━━━━━━━━━━━━━━━ 10s 0us/step

Model: "functional"

Es mucho concidero yo .

Total params: 22,852,778 (87.18 MB)

Trainable params: 1,049,994 (4.01 MB)

Non-trainable params: 21,802,784 (83.17 MB)

3. Ajuste de Hiperparámetros y Fine-tuning:

CNN cero

Intento 1

Capas convolucionales: 3 (32, 64, 128 filtros).

Capas densas: 2 (128 y 10 neuronas).

Tasa de aprendizaje: 0.0001.

Épocas de entrenamiento: 20.

Entrenamiento

Epoch 1/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 0s 694ms/step - accuracy: 0.1755 - loss: 3.0943

158/158 ━━━━━━━━━━━━━━━━━━━━ 129s 742ms/step - accuracy: 0.1757 - loss: 3.0906 - val_accuracy: 0.1775 - val_loss: 4.4247

Epoch 2/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.2922 - loss: 2.0505 - val_accuracy: 0.1919 - val_loss: 4.0960

Epoch 3/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 152ms/step - accuracy: 0.3376 - loss: 1.9528 - val_accuracy: 0.3225 - val_loss: 2.3217

Epoch 4/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.3711 - loss: 1.8437 - val_accuracy: 0.5159 - val_loss: 1.5404

Epoch 5/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.3940 - loss: 1.7784 - val_accuracy: 0.5677 - val_loss: 1.3055

Epoch 6/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.4220 - loss: 1.7162 - val_accuracy: 0.5828 - val_loss: 1.2533

Epoch 7/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.4483 - loss: 1.6532 - val_accuracy: 0.6027 - val_loss: 1.2126

Epoch 8/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 155ms/step - accuracy: 0.4544 - loss: 1.5887 - val_accuracy: 0.6361 - val_loss: 1.1142

Epoch 9/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.4701 - loss: 1.5612 - val_accuracy: 0.6314 - val_loss: 1.1300

Epoch 10/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.4652 - loss: 1.5703 - val_accuracy: 0.6202 - val_loss: 1.1578

Epoch 11/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.4809 - loss: 1.5185 - val_accuracy: 0.6417 - val_loss: 1.0854

Epoch 12/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.4952 - loss: 1.4947 - val_accuracy: 0.6513 - val_loss: 1.0911

Epoch 13/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.5002 - loss: 1.4738 - val_accuracy: 0.6537 - val_loss: 1.1066

Epoch 14/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.5229 - loss: 1.4362 - val_accuracy: 0.6473 - val_loss: 1.0800

Epoch 15/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.5301 - loss: 1.4193 - val_accuracy: 0.6497 - val_loss: 1.1092

Epoch 16/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 147ms/step - accuracy: 0.5216 - loss: 1.3885 - val_accuracy: 0.6489 - val_loss: 1.1513

Epoch 17/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.5397 - loss: 1.3448 - val_accuracy: 0.6561 - val_loss: 1.0941

Epoch 18/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.5480 - loss: 1.3540 - val_accuracy: 0.6935 - val_loss: 0.9734

Epoch 19/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.5433 - loss: 1.3442 - val_accuracy: 0.6855 - val_loss: 1.0042

Epoch 20/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.5600 - loss: 1.3179 - val_accuracy: 0.6632 - val_loss: 1.1103

Resultado

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 688ms/step

Probabilidades predichas: [[7.6634175e-01 1.5471841e-01 1.8256930e-04 2.6065083e-03 2.1045500e-02

1.6118800e-03 2.0995019e-03 2.9583112e-03 1.3356742e-03 4.7099933e-02]]

La imagen pertenece a la clase: ajedrez

Intento 2

Entrenamiento

Número total de capas: 13

Neuronas por capa: 64, 128, 256, 512 (en capas convolucionales), 512, 256 (en capas densas), 10 (capa de salida).

Tasa de aprendizaje: 0.00005

Épocas de entrenamiento: 20

Epoch 1/20

```
158/158 ━━━━━━━━━━━━━━━━━━━━ 0s 477ms/step - accuracy: 0.1268 - loss: 4.2156
158/158 ━━━━━━━━━━━━━━━━━━━━ 93s 524ms/step - accuracy: 0.1270 - loss: 4.2109 - val_accuracy: 0.1234 - val_loss: 2.8660
Epoch 2/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 47s 299ms/step - accuracy: 0.2310 - loss: 2.5200 - val_accuracy: 0.2126 - val_loss: 2.9034
Epoch 3/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 48s 303ms/step - accuracy: 0.2716 - loss: 2.1844 - val_accuracy: 0.2994 - val_loss: 2.2436
Epoch 4/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 48s 305ms/step - accuracy: 0.2886 - loss: 2.0554 - val_accuracy: 0.4021 - val_loss: 1.7081
Epoch 5/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 48s 306ms/step - accuracy: 0.3238 - loss: 1.9770 - val_accuracy: 0.5151 - val_loss: 1.4257
Epoch 6/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 49s 313ms/step - accuracy: 0.3487 - loss: 1.9247 - val_accuracy: 0.5175 - val_loss: 1.3865
Epoch 7/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 49s 307ms/step - accuracy: 0.3786 - loss: 1.8373 - val_accuracy: 0.5318 - val_loss: 1.3529
Epoch 8/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 48s 303ms/step - accuracy: 0.3924 - loss: 1.7794 - val_accuracy: 0.5318 - val_loss: 1.3455
Epoch 9/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 48s 302ms/step - accuracy: 0.4021 - loss: 1.7535 - val_accuracy: 0.5573 - val_loss: 1.2996
Epoch 10/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 48s 306ms/step - accuracy: 0.4135 - loss: 1.7215 - val_accuracy: 0.5621 - val_loss: 1.2767
Epoch 11/20
```

158/158 ━━━━━━━━━━━━━━━━━━━━ 49s 307ms/step - accuracy: 0.4196 - loss: 1.6854 - val_accuracy: 0.5820 - val_loss: 1.2770

Epoch 12/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 48s 304ms/step - accuracy: 0.4329 - loss: 1.6540 - val_accuracy: 0.5685 - val_loss: 1.2739

Epoch 13/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 47s 299ms/step - accuracy: 0.4225 - loss: 1.6532 - val_accuracy: 0.5685 - val_loss: 1.2459

Epoch 14/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 47s 299ms/step - accuracy: 0.4597 - loss: 1.5912 - val_accuracy: 0.5780 - val_loss: 1.2461

Epoch 15/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 47s 298ms/step - accuracy: 0.4526 - loss: 1.5855 - val_accuracy: 0.5780 - val_loss: 1.2477

Epoch 16/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 47s 298ms/step - accuracy: 0.4811 - loss: 1.5530 - val_accuracy: 0.5772 - val_loss: 1.2804

Epoch 17/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 47s 300ms/step - accuracy: 0.4658 - loss: 1.5597 - val_accuracy: 0.5772 - val_loss: 1.2976

Epoch 18/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 48s 300ms/step - accuracy: 0.4976 - loss: 1.5079 - val_accuracy: 0.5876 - val_loss: 1.3028

Epoch 19/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 47s 299ms/step - accuracy: 0.4917 - loss: 1.5158 - val_accuracy: 0.5900 - val_loss: 1.3056

Epoch 20/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 47s 300ms/step - accuracy: 0.5053 - loss: 1.4689 - val_accuracy: 0.5613 - val_loss: 1.4412

Resultado:

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 759ms/step

Probabilidades predichas: [[6.3904268e-01 4.6268832e-03 1.0647571e-05 1.8780770e-03 3.2737848e-01

3.3115069e-03 1.5684500e-03 1.0513433e-02 1.0284714e-02 1.3849881e-03]]

La imagen pertenece a la clase: ajedrez

Intento 3

Entrenamiento

Capas convolucionales: 3 (32, 64, 128 filtros).

Capas densas: 2 (128 y 10 neuronas).

Tasa de aprendizaje: 0.1.

Épocas de entrenamiento: 20.

Epoch 1/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 0s 430ms/step - accuracy: 0.1098 - loss: 48.8317

158/158 ━━━━━━━━━━━━━━━━━━━━ 82s 472ms/step - accuracy: 0.1098 - loss: 48.6214 - val_accuracy: 0.0223 - val_loss: 166.6405

Epoch 2/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 26s 162ms/step - accuracy: 0.0977 - loss: 2.3019 - val_accuracy: 0.1234 - val_loss: 2.3048

Epoch 3/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 25s 158ms/step - accuracy: 0.1127 - loss: 2.3010 - val_accuracy: 0.1059 - val_loss: 2.3014

Epoch 4/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 25s 156ms/step - accuracy: 0.1055 - loss: 2.3039 - val_accuracy: 0.1059 - val_loss: 2.3041

Epoch 5/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 154ms/step - accuracy: 0.1159 - loss: 2.3031 - val_accuracy: 0.0979 - val_loss: 2.3127

Epoch 6/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 154ms/step - accuracy: 0.1048 - loss: 2.3063 - val_accuracy: 0.1115 - val_loss: 2.2974

Epoch 7/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.1043 - loss: 2.3060 - val_accuracy: 0.1234 - val_loss: 2.2938

Epoch 8/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.1116 - loss: 2.2990 - val_accuracy: 0.1075 - val_loss: 2.2944

Epoch 9/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.1050 - loss: 2.3047 - val_accuracy: 0.1099 - val_loss: 2.3004

Epoch 10/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 148ms/step - accuracy: 0.1142 - loss: 2.3034 - val_accuracy: 0.1234 - val_loss: 2.3050

Epoch 11/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 147ms/step - accuracy: 0.1099 - loss: 2.3045 - val_accuracy: 0.1234 - val_loss: 2.2956

Epoch 12/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.1085 - loss: 2.3067 - val_accuracy: 0.1234 - val_loss: 2.2943

Epoch 13/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.1182 - loss: 2.2994 - val_accuracy: 0.1234 - val_loss: 2.3014

Epoch 14/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.1163 - loss: 2.3072 - val_accuracy: 0.1107 - val_loss: 2.2988

Epoch 15/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.1218 - loss: 2.3008 - val_accuracy: 0.1107 - val_loss: 2.3037

Epoch 16/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.1125 - loss: 2.3001 - val_accuracy: 0.1107 - val_loss: 2.2970

Epoch 17/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.1152 - loss: 2.2991 - val_accuracy: 0.1059 - val_loss: 2.3121

Epoch 18/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.1106 - loss: 2.3050 - val_accuracy: 0.1099 - val_loss: 2.3032

Epoch 19/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.1112 - loss: 2.2965 - val_accuracy: 0.1115 - val_loss: 2.3006

Epoch 20/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.1214 - loss: 2.2987 - val_accuracy: 0.1075 - val_loss: 2.2954

Resultado:

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step

Probabilidades predichas: [[0.09412367 0.07104202 0.11214792 0.07394501 0.09356966 0.09034933

0.11499183 0.11127781 0.12282662 0.11572617]]

Traceback (most recent call last):

File "d:\Trabajos_escuela\IA2\Seminario\Programas\Practica2_CNN\predecir.py", line 31, in <module>

predicted_class_label = class_labels[predicted_class_index]

~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^

No sabe a que clase pertenece y da error por el poco Accuracy

Intento 4:

Entrenamiento:

Capas convolucionales: 3 (32, 64, 128 filtros).

Capas densas: 2 (128 y 10 neuronas).

Tasa de aprendizaje: 0.0001.

Épocas de entrenamiento: 30.

Epoch 1/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 0s 420ms/step - accuracy: 0.1666 - loss: 3.1708

158/158 ━━━━━━━━━━━━━━━━━━━━ 75s 461ms/step - accuracy: 0.1669 - loss: 3.1668 - val_accuracy: 0.1075 - val_loss: 2.7652

Epoch 2/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.2877 - loss: 2.0327 - val_accuracy: 0.2476 - val_loss: 2.3415
~~~~~~~~~~~~

Epoch 3/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.3577 - loss: 1.8724 - val_accuracy: 0.4379 - val_loss: 1.6226

Epoch 4/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.3808 - loss: 1.8162 - val_accuracy: 0.5669 - val_loss: 1.2827

Epoch 5/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.4169 - loss: 1.7113 - val_accuracy: 0.5852 - val_loss: 1.2048

Epoch 6/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.4273 - loss: 1.6540 - val_accuracy: 0.5947 - val_loss: 1.1344

Epoch 7/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.4506 - loss: 1.6239 - val_accuracy: 0.6290 - val_loss: 1.0963

Epoch 8/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 153ms/step - accuracy: 0.4515 - loss: 1.6124 - val_accuracy: 0.6202 - val_loss: 1.1001

Epoch 9/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 149ms/step - accuracy: 0.4724 - loss: 1.5567 - val_accuracy: 0.6449 - val_loss: 1.0636

Epoch 10/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.4848 - loss: 1.5236 - val_accuracy: 0.6561 - val_loss: 1.0367

Epoch 11/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.4941 - loss: 1.4966 - val_accuracy: 0.6553 - val_loss: 1.0830

Epoch 12/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 147ms/step - accuracy: 0.5019 - loss: 1.4741 - val_accuracy: 0.6592 - val_loss: 1.0233

Epoch 13/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 147ms/step - accuracy: 0.4971 - loss: 1.4532 - val_accuracy: 0.6775 - val_loss: 1.0130

Epoch 14/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.5326 - loss: 1.4038 - val_accuracy: 0.6823 - val_loss: 1.0060

Epoch 15/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.5263 - loss: 1.3992 - val_accuracy: 0.6967 - val_loss: 0.9127

Epoch 16/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.5438 - loss: 1.3692 - val_accuracy: 0.6975 - val_loss: 0.9116

Epoch 17/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.5419 - loss: 1.3598 - val_accuracy: 0.6935 - val_loss: 0.9121

Epoch 18/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.5503 - loss: 1.3311 - val_accuracy: 0.6863 - val_loss: 0.9502

Epoch 19/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.5461 - loss: 1.3301 - val_accuracy: 0.7054 - val_loss: 0.9068

Epoch 20/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.5491 - loss: 1.3325 - val_accuracy: 0.6943 - val_loss: 0.9643

Epoch 21/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.5693 - loss: 1.3191 - val_accuracy: 0.7094 - val_loss: 0.9273

Epoch 22/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.5789 - loss: 1.2633 - val_accuracy: 0.7102 - val_loss: 0.9698

Epoch 23/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.5686 - loss: 1.2681 - val_accuracy: 0.6919 - val_loss: 0.9672

Epoch 24/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.5820 - loss: 1.2333 - val_accuracy: 0.7070 - val_loss: 0.8909

Epoch 25/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.5865 - loss: 1.2201 - val_accuracy: 0.7166 - val_loss: 0.8687

Epoch 26/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.6035 - loss: 1.1975 - val_accuracy: 0.6783 - val_loss: 1.0460

Epoch 27/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.6021 - loss: 1.1983 - val_accuracy: 0.7046 - val_loss: 0.9772

Epoch 28/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.6087 - loss: 1.1710 - val_accuracy: 0.7086 - val_loss: 0.9864

Epoch 29/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.6250 - loss: 1.1402 - val_accuracy: 0.7404 - val_loss: 0.8509

Epoch 30/30

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.6066 - loss: 1.1492 - val_accuracy: 0.7150 - val_loss: 0.9300

Intento 5

Capas convolucionales: 3 (32, 64, 128 filtros).

Capas densas: 2 (128 y 10 neuronas).

Tasa de aprendizaje: 0.00001.

Épocas de entrenamiento: 20.

Epoch 1/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 0s 432ms/step - accuracy: 0.1036 - loss: 4.6228

158/158 ━━━━━━━━━━━━━━━━━━━━ 87s 479ms/step - accuracy: 0.1036 - loss: 4.6200 - val_accuracy: 0.1266 - val_loss: 2.4567

Epoch 2/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 32s 201ms/step - accuracy: 0.1412 - loss: 3.2436 - val_accuracy: 0.1768 - val_loss: 2.2836

Epoch 3/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 27s 171ms/step - accuracy: 0.1901 - loss: 2.6147 - val_accuracy: 0.3296 - val_loss: 1.9661

Epoch 4/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.1974 - loss: 2.4402 - val_accuracy: 0.3933 - val_loss: 1.7762

Epoch 5/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.2235 - loss: 2.2820 - val_accuracy: 0.4299 - val_loss: 1.7052

Epoch 6/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.2497 - loss: 2.1863 - val_accuracy: 0.4395 - val_loss: 1.6702

Epoch 7/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.2563 - loss: 2.1475 - val_accuracy: 0.4634 - val_loss: 1.6255

Epoch 8/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.2787 - loss: 2.0943 - val_accuracy: 0.4873 - val_loss: 1.5691

Epoch 9/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.2742 - loss: 2.0698 - val_accuracy: 0.4912 - val_loss: 1.5505

Epoch 10/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.3168 - loss: 1.9960 - val_accuracy: 0.4960 - val_loss: 1.5201

Epoch 11/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.3074 - loss: 1.9919 - val_accuracy: 0.5008 - val_loss: 1.4905

Epoch 12/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.3280 - loss: 1.9498 - val_accuracy: 0.5207 - val_loss: 1.4605

Epoch 13/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.3367 - loss: 1.9510 - val_accuracy: 0.5263 - val_loss: 1.4244

Epoch 14/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.3246 - loss: 1.9492 - val_accuracy: 0.5366 - val_loss: 1.3934

Epoch 15/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.3574 - loss: 1.8934 - val_accuracy: 0.5334 - val_loss: 1.3700

Epoch 16/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 155ms/step - accuracy: 0.3656 - loss: 1.8609 - val_accuracy: 0.5478 - val_loss: 1.3669

Epoch 17/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.3590 - loss: 1.8754 - val_accuracy: 0.5470 - val_loss: 1.3416

Epoch 18/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 151ms/step - accuracy: 0.3470 - loss: 1.8670 - val_accuracy: 0.5677 - val_loss: 1.3305

Epoch 19/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.3643 - loss: 1.8517 - val_accuracy: 0.5661 - val_loss: 1.3012

Epoch 20/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.3808 - loss: 1.8247 - val_accuracy: 0.5621 - val_loss: 1.2942

Intento 6:

Capas convolucionales: 3 (32, 64, 128 filtros).

Capas densas: 2 (128 y 10 neuronas).

Tasa de aprendizaje: 0.001.

Épocas de entrenamiento: 20.

Epoch 1/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step - accuracy: 0.2315 - loss: 2.7354

158/158 ━━━━━━━━━━━━━━━━━━━━ 27s 155ms/step - accuracy: 0.2318 - loss: 2.7321 - val_accuracy: 0.1473 - val_loss: 4.5405

Epoch 2/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.3561 - loss: 1.8841 - val_accuracy: 0.2898 - val_loss: 2.6907

Epoch 3/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 152ms/step - accuracy: 0.3898 - loss: 1.8147 - val_accuracy: 0.3607 - val_loss: 1.9633

Epoch 4/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 152ms/step - accuracy: 0.4167 - loss: 1.7221 - val_accuracy: 0.5494 - val_loss: 1.3324

Epoch 5/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 153ms/step - accuracy: 0.4436 - loss: 1.6470 - val_accuracy: 0.5390 - val_loss: 1.3928

Epoch 6/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 152ms/step - accuracy: 0.4599 - loss: 1.5760 - val_accuracy: 0.5637 - val_loss: 1.3085

Epoch 7/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 152ms/step - accuracy: 0.4861 - loss: 1.5654 - val_accuracy: 0.5597 - val_loss: 1.2477

Epoch 8/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 25s 155ms/step - accuracy: 0.4927 - loss: 1.5044 - val_accuracy: 0.6194 - val_loss: 1.0801

Epoch 9/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 153ms/step - accuracy: 0.5050 - loss: 1.4701 - val_accuracy: 0.6027 - val_loss: 1.1388

Epoch 10/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 25s 155ms/step - accuracy: 0.5151 - loss: 1.4387 - val_accuracy: 0.6417 - val_loss: 1.0513

```
Epoch 11/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 154ms/step - accuracy: 0.5373 - loss: 1.3761 - val_accuracy: 0.6409 - val_loss: 1.0445
Epoch 12/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 154ms/step - accuracy: 0.5518 - loss: 1.3244 - val_accuracy: 0.6696 - val_loss: 0.9553
Epoch 13/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 153ms/step - accuracy: 0.5464 - loss: 1.3461 - val_accuracy: 0.5518 - val_loss: 1.2577
Epoch 14/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 154ms/step - accuracy: 0.5635 - loss: 1.2642 - val_accuracy: 0.7349 - val_loss: 0.8581
Epoch 15/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 153ms/step - accuracy: 0.5850 - loss: 1.2289 - val_accuracy: 0.6545 - val_loss: 1.0413
Epoch 16/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 25s 157ms/step - accuracy: 0.5734 - loss: 1.2577 - val_accuracy: 0.7452 - val_loss: 0.8205
Epoch 17/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 150ms/step - accuracy: 0.5942 - loss: 1.1976 - val_accuracy: 0.7381 - val_loss: 0.8210
Epoch 18/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.6127 - loss: 1.1633 - val_accuracy: 0.7341 - val_loss: 0.7908
Epoch 19/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.6054 - loss: 1.1883 - val_accuracy: 0.6847 - val_loss: 0.9450
Epoch 20/20
158/158 ━━━━━━━━━━━━━━━━━━━━ 25s 159ms/step - accuracy: 0.6032 - loss: 1.1786 - val_accuracy: 0.7365 - val_loss: 0.8508
```

Transfer Learning

Epoch 1/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 244s 1s/step - accuracy: 0.2280 - loss: 2.2084 - val_accuracy: 0.6943 - val_loss: 1.2393

Epoch 2/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 232s 1s/step - accuracy: 0.5271 - loss: 1.4636 - val_accuracy: 0.7691 - val_loss: 0.9072

Epoch 3/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 232s 1s/step - accuracy: 0.6132 - loss: 1.1923 - val_accuracy: 0.8145 - val_loss: 0.7422

Epoch 4/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 231s 1s/step - accuracy: 0.6602 - loss: 1.0719 - val_accuracy: 0.8232 - val_loss: 0.6527

Epoch 5/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 231s 1s/step - accuracy: 0.6865 - loss: 0.9565 - val_accuracy: 0.8471 - val_loss: 0.5575

Epoch 6/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 232s 1s/step - accuracy: 0.7239 - loss: 0.8725 - val_accuracy: 0.8519 - val_loss: 0.5492

Epoch 7/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 232s 1s/step - accuracy: 0.7393 - loss: 0.8303 - val_accuracy: 0.8607 - val_loss: 0.5063

Epoch 8/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 235s 1s/step - accuracy: 0.7584 - loss: 0.7805 - val_accuracy: 0.8670 - val_loss: 0.4832

Epoch 9/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 233s 1s/step - accuracy: 0.7702 - loss: 0.7261 - val_accuracy: 0.8782 - val_loss: 0.4441

Epoch 10/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 233s 1s/step - accuracy: 0.7814 - loss: 0.6917 - val_accuracy: 0.8806 - val_loss: 0.4430

Epoch 1/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 746s 5s/step - accuracy: 0.7898 - loss: 0.6596 - val_accuracy: 0.8854 - val_loss: 0.3550 - learning_rate: 1.0000e-05

Epoch 2/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 744s 5s/step - accuracy: 0.8295 - loss: 0.5122 - val_accuracy: 0.8925 - val_loss: 0.3153 - learning_rate: 1.0000e-05

Epoch 3/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 742s 5s/step - accuracy: 0.8824 - loss: 0.3646 - val_accuracy: 0.9188 - val_loss: 0.2485 - learning_rate: 1.0000e-05

Epoch 4/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 742s 5s/step - accuracy: 0.8970 - loss: 0.3282 - val_accuracy: 0.9260 - val_loss: 0.2370 - learning_rate: 1.0000e-05

Epoch 5/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 744s 5s/step - accuracy: 0.9067 - loss: 0.3134 - val_accuracy: 0.9307 - val_loss: 0.2239 - learning_rate: 1.0000e-05

Epoch 6/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 744s 5s/step - accuracy: 0.9234 - loss: 0.2453 - val_accuracy: 0.9092 - val_loss: 0.2567 - learning_rate: 1.0000e-05

Epoch 7/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 743s 5s/step - accuracy: 0.9332 - loss: 0.2046 - val_accuracy: 0.9379 - val_loss: 0.2146 - learning_rate: 1.0000e-05

Epoch 8/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 742s 5s/step - accuracy: 0.9394 - loss: 0.1979 - val_accuracy: 0.9355 - val_loss: 0.2164 - learning_rate: 1.0000e-05

Epoch 9/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 740s 5s/step - accuracy: 0.9509 - loss: 0.1523 - val_accuracy: 0.9387 - val_loss: 0.2143 - learning_rate: 1.0000e-05

Epoch 10/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 742s 5s/step - accuracy: 0.9555 - loss: 0.1466 - val_accuracy: 0.9427 - val_loss: 0.2103 - learning_rate: 1.0000e-05

Epoch 11/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 743s 5s/step - accuracy: 0.9641 - loss: 0.1127 - val_accuracy: 0.9395 - val_loss: 0.2147 - learning_rate: 1.0000e-05

Epoch 12/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 744s 5s/step - accuracy: 0.9711 - loss: 0.0995 - val_accuracy: 0.9427 - val_loss: 0.1857 - learning_rate: 1.0000e-05

Evaluación y Métricas:

CNN desde cero

Epoch 1/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 146s 872ms/step - accuracy: 0.1778 - loss: 3.1778 - val_accuracy: 0.1234 - val_loss: 4.4577

Epoch 2/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 73s 444ms/step - accuracy: 0.2985 - loss: 2.0352 - val_accuracy: 0.1409 - val_loss: 4.0426

Epoch 3/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 58s 351ms/step - accuracy: 0.3530 - loss: 1.9063 - val_accuracy: 0.3702 - val_loss: 2.0613

Epoch 4/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 23s 144ms/step - accuracy: 0.3852 - loss: 1.8252 - val_accuracy: 0.5494 - val_loss: 1.3203

Epoch 5/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 25s 155ms/step - accuracy: 0.4095 - loss: 1.7200 - val_accuracy: 0.5844 - val_loss: 1.1983

Epoch 6/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 31s 191ms/step - accuracy: 0.4139 - loss: 1.7078 - val_accuracy: 0.6282 - val_loss: 1.1148

Epoch 7/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 29s 179ms/step - accuracy: 0.4401 - loss: 1.6385 - val_accuracy: 0.6393 - val_loss: 1.0803

Epoch 8/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 149ms/step - accuracy: 0.4436 - loss: 1.6339 - val_accuracy: 0.6154 - val_loss: 1.1181

Epoch 9/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 147ms/step - accuracy: 0.4669 - loss: 1.5258 - val_accuracy: 0.6354 - val_loss: 1.0971

Epoch 10/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 146ms/step - accuracy: 0.4661 - loss: 1.5594 - val_accuracy: 0.6377 - val_loss: 1.1010

Epoch 11/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 146ms/step - accuracy: 0.4889 - loss: 1.5176 - val_accuracy: 0.6521 - val_loss: 1.0516

Epoch 12/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 146ms/step - accuracy: 0.4981 - loss: 1.4813 - val_accuracy: 0.6632 - val_loss: 1.0280

Epoch 13/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 147ms/step - accuracy: 0.5035 - loss: 1.4692 - val_accuracy: 0.6561 - val_loss: 1.0697

Epoch 14/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 147ms/step - accuracy: 0.5241 - loss: 1.4184 - val_accuracy: 0.6521 - val_loss: 1.0872

Epoch 15/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 24s 146ms/step - accuracy: 0.5276 - loss: 1.4109 - val_accuracy: 0.6815 - val_loss: 0.9590

Epoch 16/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 59s 375ms/step - accuracy: 0.5253 - loss: 1.3799 - val_accuracy: 0.6815 - val_loss: 1.0082

Epoch 17/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 98s 588ms/step - accuracy: 0.5517 - loss: 1.3243 - val_accuracy: 0.6871 - val_loss: 0.9878

Epoch 18/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 26s 161ms/step - accuracy: 0.5545 - loss: 1.3282 - val_accuracy: 0.6887 - val_loss: 0.9605

Epoch 19/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 25s 153ms/step - accuracy: 0.5594 - loss: 1.3032 - val_accuracy: 0.7118 - val_loss: 0.9523

Epoch 20/20

158/158 ━━━━━━━━━━━━━━━━━━━━ 25s 152ms/step - accuracy: 0.5691 - loss: 1.3097 - val_accuracy: 0.6807 - val_loss: 1.1115

Model: "sequential"

```
| Layer (type)                  | Output Shape           | Param # |
| conv2d (Conv2D)               | (None, 62, 62, 32)     |     896 |
| batch_normalization           | (None, 62, 62, 32)     |     128 |
| (BatchNormalization)          |                        |         |
| max_pooling2d (MaxPooling2D)  | (None, 31, 31, 32)     |       0 |
| dropout (Dropout)             | (None, 31, 31, 32)     |       0 |
| conv2d_1 (Conv2D)             | (None, 29, 29, 64)     |  18,496 |
| batch_normalization_1         | (None, 29, 29, 64)     |     256 |
| (BatchNormalization)          |                        |         |
| max_pooling2d_1 (MaxPooling2D)| (None, 14, 14, 64)     |       0 |
```

dropout_1 (Dropout)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 12, 12, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 12, 12, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 128)	0
dropout_2 (Dropout)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 128)	589,952
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 2,055,264 (7.84 MB)

Trainable params: 684,938 (2.61 MB)

Non-trainable params: 448 (1.75 KB)

Optimizer params: 1,369,878 (5.23 MB)

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

40/40 ━━━━━━━━━━━━━━━━━━━━ 3s 70ms/step

Reporte de clasificación para el conjunto de validación:

```
              precision    recall  f1-score   support

     ajedrez       0.14      0.09      0.11        93
  baloncesto       0.07      0.04      0.05        96
       boxeo       0.07      0.04      0.05       138
     disparo       0.00      0.00      0.00       104
     esgrima       0.15      0.15      0.15       123
    formula1       0.09      0.18      0.12       133
      futbol       0.11      0.12      0.11       155
      hockey       0.10      0.12      0.11       139
    natacion       0.08      0.10      0.09       135
       tenis       0.12      0.12      0.12       140

    accuracy                           0.10      1256
   macro avg       0.09      0.10      0.09      1256
weighted avg       0.09      0.10      0.09      1256
```

Matriz de Confusión:

```
[[ 8  4  9  0 10 13 15 13 15  6]
 [ 8  4  3  3  4 19 11 17 12 15]
 [ 3  9  6  6 16 30 18 19 13 18]
```

[3 3 11 0 5 28 9 15 18 12]

[5 6 6 2 19 21 20 18 14 12]

[9 5 7 3 13 24 21 16 16 19]

[2 10 16 4 11 44 18 21 19 10]

[4 4 11 4 18 26 19 17 21 15]

[9 5 12 2 15 26 21 15 13 17]

[6 9 6 6 13 32 19 18 14 17]]

Pérdida en entrenamiento: 1.4034, Precisión en entrenamiento: 0.6083

Pérdida en validación: 1.1012, Precisión en validación: 0.6855

Transfer Learning

Epoch 1/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 227s 1s/step - accuracy: 0.2291 - loss: 2.1703 - val_accuracy: 0.6720 - val_loss: 1.2235

Epoch 2/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 229s 1s/step - accuracy: 0.5225 - loss: 1.4450 - val_accuracy: 0.7747 - val_loss: 0.8626

Epoch 3/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 233s 1s/step - accuracy: 0.6063 - loss: 1.2061 - val_accuracy: 0.8225 - val_loss: 0.7109

Epoch 4/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 235s 1s/step - accuracy: 0.6577 - loss: 1.0612 - val_accuracy: 0.8416 - val_loss: 0.6205

Epoch 5/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 235s 1s/step - accuracy: 0.6947 - loss: 0.9471 - val_accuracy: 0.8527 - val_loss: 0.5770

Epoch 6/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 238s 2s/step - accuracy: 0.7185 - loss: 0.9033 - val_accuracy: 0.8591 - val_loss: 0.5376

Epoch 7/10

```
158/158 ━━━━━━━━━━━━━━━━━━━━ 235s 1s/step - accuracy: 0.7239 - loss: 0.8506 - val_accuracy: 0.8774 - val_loss: 0.4881
Epoch 8/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 235s 1s/step - accuracy: 0.7536 - loss: 0.7734 - val_accuracy: 0.8591 - val_loss: 0.4899
Epoch 9/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 235s 1s/step - accuracy: 0.7557 - loss: 0.7570 - val_accuracy: 0.8694 - val_loss: 0.4615
Epoch 10/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 235s 1s/step - accuracy: 0.7948 - loss: 0.6792 - val_accuracy: 0.8678 - val_loss: 0.4536
Epoch 1/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 444s 3s/step - accuracy: 0.7954 - loss: 0.6315 - val_accuracy: 0.8973 - val_loss: 0.3178 - learning_rate: 1.0000e-05
Epoch 2/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 442s 3s/step - accuracy: 0.8568 - loss: 0.4505 - val_accuracy: 0.9029 - val_loss: 0.2928 - learning_rate: 1.0000e-05
Epoch 3/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 441s 3s/step - accuracy: 0.8678 - loss: 0.4019 - val_accuracy: 0.9108 - val_loss: 0.2585 - learning_rate: 1.0000e-05
Epoch 4/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 442s 3s/step - accuracy: 0.8777 - loss: 0.3761 - val_accuracy: 0.9236 - val_loss: 0.2321 - learning_rate: 1.0000e-05
Epoch 5/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 441s 3s/step - accuracy: 0.9049 - loss: 0.2916 - val_accuracy: 0.9172 - val_loss: 0.2679 - learning_rate: 1.0000e-05
Epoch 6/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 442s 3s/step - accuracy: 0.9196 - loss: 0.2448 - val_accuracy: 0.9220 - val_loss: 0.2401 - learning_rate: 1.0000e-05
Epoch 7/10
158/158 ━━━━━━━━━━━━━━━━━━━━ 444s 3s/step - accuracy: 0.9306 - loss: 0.2186 - val_accuracy: 0.9204 - val_loss: 0.2345 - learning_rate: 1.0000e-05
```

Epoch 8/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 442s 3s/step - accuracy: 0.9513 - loss: 0.1617 - val_accuracy: 0.9283 - val_loss: 0.2113 - learning_rate: 5.0000e-06

Epoch 9/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 444s 3s/step - accuracy: 0.9518 - loss: 0.1516 - val_accuracy: 0.9331 - val_loss: 0.2234 - learning_rate: 5.0000e-06

Epoch 10/10

158/158 ━━━━━━━━━━━━━━━━━━━━ 442s 3s/step - accuracy: 0.9510 - loss: 0.1627 - val_accuracy: 0.9283 - val_loss: 0.2428 - learning_rate: 5.0000e-06

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

Evaluación del modelo inicial:

40/40 ━━━━━━━━━━━━━━━━━━━━ 46s 1s/step

Reporte de clasificación para el modelo inicial:

	precision	recall	f1-score	support
ajedrez	0.05	0.05	0.05	93
baloncesto	0.06	0.05	0.06	96
boxeo	0.08	0.09	0.09	138
disparo	0.11	0.11	0.11	104
esgrima	0.06	0.06	0.06	123
formula1	0.10	0.10	0.10	133
futbol	0.14	0.14	0.14	155
hockey	0.07	0.07	0.07	139
natacion	0.10	0.10	0.10	135
tenis	0.14	0.14	0.14	140

accuracy			0.10	1256
macro avg	0.09	0.09	0.09	1256
weighted avg	0.10	0.10	0.10	1256

Matriz de Confusión para el modelo inicial:

```
[[ 5  6 11  8 16  8 11 10 10  8]
 [ 6  5  8  5 10 10 15 11 14 12]
 [12 15 13 14 15 10 19 15 14 11]
 [ 5  7  8 11 15 13 10 13  9 13]
 [ 8 10 23  7  7  8 14 17 14 15]
 [12 11 17 12 13 13 13 12 19 11]
 [11  9 20  8 11 18 22 21 13 22]
 [12  5 20 11 11 15 25 10 12 18]
 [14  4 18  7 18 15 22 12 14 11]
 [ 9  7 18 15 10 20 11 14 16 20]]
```

Pérdida en entrenamiento (inicial): 0.0581, Precisión en entrenamiento: 0.9837

Pérdida en validación (inicial): 0.2361, Precisión en validación: 0.9260

Evaluación del modelo con fine-tuning:

40/40 ━━━━━━━━━━━━━━━━━━━━ 47s 1s/step

Reporte de clasificación para el modelo con fine-tuning:

	precision	recall	f1-score	support
ajedrez	0.09	0.09	0.09	93
baloncesto	0.04	0.03	0.03	96
boxeo	0.11	0.13	0.12	138
disparo	0.13	0.12	0.12	104

esgrima	0.14	0.15	0.14	123
formula1	0.11	0.11	0.11	133
futbol	0.15	0.15	0.15	155
hockey	0.15	0.15	0.15	139
natacion	0.10	0.10	0.10	135
tenis	0.12	0.12	0.12	140
accuracy			0.12	1256
macro avg	0.11	0.11	0.11	1256
weighted avg	0.12	0.12	0.12	1256

Matriz de Confusión para el modelo con fine-tuning:

```
[[ 8  8 16  8  8  8  9 10  9  9]
 [ 9  3 10  8  6  6 17 10 15 12]
 [ 7 14 18  8 14 14 19 14 14 16]
 [ 6  5 19 12  9  6 14 11 13  9]
 [11  8 19  9 18 12 11 14  9 12]
 [12  9 17 12 12 14 10 14 18 15]
 [10  7 19 15 14 17 24 11 19 19]
 [ 6  5 17  8 15 14 16 21 19 18]
 [ 9 11 14  8  9 21 21 15 13 14]
 [13 10 13  6 21 20 17 16  7 17]]
```

Pérdida en entrenamiento (fine-tuning): 0.0600, Precisión en entrenamiento: 0.9847

Pérdida en validación (fine-tuning): 0.2543, Precisión en validación: 0.9275

Comprobación con las imágenes de validación

Modelo de predicción utilizado: 'intento5_deportes.h5'

Imagen ajedrez

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 381ms/step

Probabilidades predichas: [[7.0239538e-01 9.8915741e-02 7.3395792e-04 2.3882557e-02 1.3948210e-01

1.0196291e-02 2.5506886e-03 1.6070386e-02 6.3668413e-04 5.1362277e-03]]

La imagen pertenece a la clase: ajedrez

Imagen basket

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step

Probabilidades predichas: [[7.5131334e-02 4.7606602e-01 4.0907357e-04 1.2864300e-02 5.2681779e-03

5.6504674e-02 1.3834948e-02 2.8174758e-01 1.9322714e-04 7.7980667e-02]]

La imagen pertenece a la clase: basketball

Imagen boxeo

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step

Probabilidades predichas: [[4.4956811e-02 7.6526128e-02 6.7588538e-01 1.3771431e-02 9.6987211e-04

2.5693083e-02 6.2941876e-04 4.8992880e-02 8.3016597e-02 2.9558433e-02]]

La imagen pertenece a la clase: boxeo

Imagen disparo

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step

Probabilidades predichas: [[0.03193022 0.00689876 0.00565208 0.09683515 0.08662546 0.6362022

0.00464861 0.01132697 0.00259895 0.11728156]]

La imagen pertenece a la clase: disparo

Imagen esgrima

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 167ms/step

Probabilidades predichas: [[0.00496225 0.00514487 0.13052835 0.01475009 0.70610607 0.01538486

0.00380396 0.01757892 0.00090857 0.10083199]]

La imagen pertenece a la clase: esgrima

Imagen formula1

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 174ms/step

Probabilidades predichas: [[3.4874897e-03 2.6660541e-01 1.8414884e-04 3.7606834e-03 4.5155492e-03

7.1122301e-01 1.1300070e-03 5.8196681e-03 4.6494511e-06 3.2693562e-03]]

La imagen pertenece a la clase: formula1

Imagen futbol

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 161ms/step

Probabilidades predichas: [[3.2778839e-06 3.7589452e-05 1.6804714e-07 7.0953846e-04 4.0691284e-05

4.0763439e-04 5.5138546e-01 2.0027715e-04 1.7355671e-05 4.4719800e-01]]

La imagen pertenece a la clase: futbol

Imagen hockey

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 145ms/step

Probabilidades predichas: [[3.0483912e-05 1.1884323e-04 2.0921423e-06 1.0952361e-04 1.3848701e-02

3.6716792e-03 1.6947313e-05 9.8210335e-01 1.0923028e-06 9.7316020e-05]]

La imagen pertenece a la clase: hockey

Imagen natación

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step

Probabilidades predichas: [[1.06089935e-10 1.13302336e-11 2.07681823e-11 8.97392230e-11

5.62003361e-11 5.34922162e-10 1.74533391e-11 5.48271151e-10

1.00000000e+00 3.91885955e-08]]

La imagen pertenece a la clase: natación

Imagen tenis

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 173ms/step

Probabilidades predichas: [[8.1049817e-05 1.8470371e-03 2.1087060e-04 4.9962853e-03 5.7446185e-05

3.9587095e-03 9.3895290e-04 2.4944218e-03 4.3482777e-01 5.5058748e-01]]

La imagen pertenece a la clase: tenis

Conclusiones y observaciones

Para concluir debo decir que me gusto mucho esta práctica, me pude dar una idea mejor de lo que es una CNN y como puede servir practicar y saber que quieres hacer (refiriéndome a que código y que parámetros utilizar dependiendo a situación), también quiero resaltar que mi computadora trabajo mucho en algunos entrenamientos, también dependía mucho de los parámetros (por lógica), pero siento que se logro demostrar de manera correcta la practica requerida.

Referencias

1. Galvis Chacón, J. A. (n.d.). *Machine learning: Redes Neuronales Convolucionales*. Universidad de Guadalajara. Centro Universitario de Ciencias Exactas e Ingenierías. Recuperado de https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

2. Towards ML. (2018). *Deep Learning Series P2: Understanding Convolutional Neural Networks*. Recuperado de https://towardsml.wordpress.com/2018/10/16/deep-learning-series-p2-understanding-convolutional-neural-networks/

3. DataScientest. (n.d.). *¿Qué es el Transfer Learning?*. Recuperado de https://datascientest.com/es/que-es-el-transfer-learning

4. NNabla. (n.d.). *ImageNet models*. Recuperado de https://nnabla.readthedocs.io/en/latest/python/api/models/imagenet.html