DESARROLLO DE MODELOS DE IA PARA RECONOCIMIENTO DE ANIMALES

WILDDETECT

NOMBRE: JOSE LUIS MAYO CALVO

CURSO: 2º DAM

CENTRO: IES RIBERA DE CASTILLA

CONTENIDO DE LA MEMORIA

INTRODUCCIÓN

El proyecto WildDetect, al ser realizado mediante un dron, con el uso de cámaras fotográficas y cámara térmica y ser este sobrevolado a unos 50 metros de altura, los animales de minúsculo tamaño podrán ser captados por la cámara térmica pero no por la cámara fotográfica con la definición requerida para poder ser reconocidos por el modelo entrenado, por lo que, de los animales existentes en la fauna de la comunidad, el modelo reconocerá algunos de ellos.

WildDetect es un proyecto de investigación que consta de la distinción de los distintos animales más comunes del territorio autonómico, como pueden ser perros, gatos, cerdos, ciervos, jabalís o lobos, entre muchos otros. Un total de 19 especies distintas de animales, que, el modelo de IA es capaz de reconocer.

Este modelo es ejecutado y creado con la herramienta de código abierto Jupyter, y organizado por celdas, en el que se utilizan librerías externas como keras o modelos preentrenados para la visualización y reconocimiento de imágenes como el VGG16.

PRIMER MODELO IA DE RECONOCIMIENTO ANIMAL

Analizamos el código para ver las herramientas utilizadas para el entrenamiento del modelo y sus posteriores resultados, así como la máxima eficacia conseguida con el mismo:

```
import os
import cv2
import random
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras import backend as K
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.applications import VGG16
import shutil
from tensorflow.keras.regularizers import l2
```

Primero, empezamos importando todas las librerías externas necesarias y modelos, como keras o tensorflow, o modelos preentrenados VGG16, especializado en procesamiento de imágenes, así como la detección de objetos en ellas.

Entre las librerías y modelos más destacados, podemos encontrar:

- El modelo OS, permite interactuar con el sistema operativo, así como hacer distintas operaciones con directorios y archivos (crear, eliminar o modificar)
- La librería OpenCV2, de código abierto, utilizada para el procesamiento de imágenes o videos (detección de objetos, análisis de imágenes…).
- El modelo random, que sirve para poder generar un número aleatorio
- La librería matplotlib, utilizada para generar gráficos, con la finalidad de poder ver el resultado del entrenamiento de una manera más visual.
- La librería PIL, para el procesamiento de imágenes (abrir, manipulación o guardar imágenes).
- Librería numPy, proporciona estructuras de datos como arrays y funciones para realizar diversas operaciones matemáticas.

- Librería tensorflow, utilizada para matching learning, concretamente Deep learning, utilizado mucho para el entrenamiento de modelos con imágenes.
- El optimizador Adam para redes neuronales, utilizado para los entrenamientos profundos que permite adaptar la tasa de aprendiza en función de las necesidades del modelo.
- Sequential es la forma de definir un modelo apilando sus capas de manera secuencial (la salida de una capa es la entrada de la siguiente), haciendo el modelo más simple y eficiente.
- Layers es la librería que importa las diferentes capas utilizadas en el entrenamiento (DropOut, Flatten, Dense y Activation).
- Herramienta TensorBoard, utilizado para visualizar el entrenamiento de los modelos, viendo los balances de precisión y perdida durante el mismo.
- Función backend, que permite realizar las operaciones y funciones básicas que se utilizan durante el entrenamiento de los modelos.
- ModelCheckPoint, permite guardar el entrenamiento durante el paso de x épocas, con la finalidad de perder información y ahorrar tiempo, pudiendo empezar un entrenamiento futuro a partir de la época guardada.
- EarlyStopping para el entrenamiento si los valores de precisión y perdida no mejoran, evitando el sobreajuste de los entrenamientos.
- ReduceRLOnPlateau: reduce la tasa de aprendizaje cuando el entrenamiento no mejora, haciendo que se memoricen parámetros más específicos y aumentando la mejora.
- El modelo VGG16 es un modelo preentrenado de procesamiento de imágenes que nos permitirá avanzar en el procesamiento de las distintas imágenes.
- Librería shutil permite realizar operaciones de alto nivel con archivos o directorios (copiar, mover, eliminar o editar)
- La función L2 es una técnica de regularización que penaliza los grandes valores de los pesos en la red neuronal, evitando así el sobreajuste del modelo.

```
# Limpiar la sesión de Keras
K.clear_session()

# Directorios de imágenes
datos_entrenamiento = '../Entrenamiento'
datos_validacion = '../Validacion'

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="./logs", histogram_freq=1)
```

Primero, limpiamos y eliminamos todos los procesos que keras esté realizando, con la finalidad de que no se acumulen modelos en memoria, lo que produciría una reducción en el rendimiento y un aumento innecesario de carga en la memoria.

Posteriormente, definimos la ruta donde se encuentran los datos (set de imágenes) de entrenamiento y validación, para que el modelo pueda obtener esas imágenes y entrenar de manera correcta.

Por último, hacemos que se pueda visualizar el entrenamiento del modelo en TensorBoard y definimos la carpeta donde se guardan los logs del entrenamiento y definimos que se registren los histogramas y pesos del modelo cada época, para evitar la pérdida de información.

```python
# Función para redimensionar sin distorsión, manteniendo la relación de aspecto
def redimensionar_manteniendo_relacion_aspecto(img, tamaño_max=224):
    altura, ancho = img.shape[:2]
    factor_escala = tamaño_max / float(max(altura, ancho))

    nuevo_ancho = int(ancho * factor_escala)
    nueva_altura = int(altura * factor_escala)

    # Redimensionar la imagen con una interpolación de alta calidad
    img_redimensionada = cv2.resize(img, (nuevo_ancho, nueva_altura), interpolation=cv2.INTER_AREA)

    # Crear una imagen de fondo cuadrada (100x100) con color negro
    fondo = np.zeros((tamaño_max, tamaño_max, 3), dtype=np.uint8)

    # Calcular las coordenadas para centrar la imagen en el fondo
    x_offset = (tamaño_max - nuevo_ancho) // 2
    y_offset = (tamaño_max - nueva_altura) // 2

    # Copiar la imagen redimensionada en el centro del fondo negro
    fondo[y_offset:y_offset + nueva_altura, x_offset:x_offset + nuevo_ancho] = img_redimensionada

    return fondo
```

El objetivo de esta función es redimensionar las imágenes en cuadro negro de 224x224 píxeles, haciendo que todas las imágenes tengan la misma dimensión, pero, evitando que, al tener la misma dimensión, la imagen se difumine y el modelo no pueda captar las características de esta de forma correcta.

De esta forma hacemos que las imágenes tengan la misma dimensión (siendo esto muy conveniente para el modelo), pero, evitando que la imagen se deforme, produciendo unos parámetros de la imagen difusos para el modelo.

Primero, obtenemos las dimensiones reales de la imagen y calculamos el factor escala de la imagen, es decir, el valor que se utiliza para redimensionar la imagen a partes iguales y evitar deformaciones en la misma, produciendo que el modelo aprenda de manera errónea dichos parámetros. Una vez obtenido el factor escala, calculamos las nuevas dimensiones de la imagen y la redimensionamos con la librería OpenCV2 y asegurándonos de que lo hace a alta calidad con la interpolación INTER_AREA.

Después, creamos el fondo negro sobre el que insertaremos la imagen, y calculamos las coordenadas de dicho fondo para poner la imagen en el centro. Una vez calculado, copiamos la imagen dentro del fondo negro y la devolvemos.

```python
# Mover todas las imágenes de validación a entrenamiento al inicio
for clase in os.listdir(datos_validacion):
    clase_validacion = os.path.join(datos_validacion, clase)
    clase_entrenamiento = os.path.join(datos_entrenamiento, clase)

    if not os.path.exists(clase_entrenamiento):
        os.makedirs(clase_entrenamiento)

    if os.path.isdir(clase_validacion):
        imagenes = os.listdir(clase_validacion)
        for imagen in imagenes:
            origen = os.path.join(clase_validacion, imagen)
            destino = os.path.join(clase_entrenamiento, imagen)

            if not os.path.exists(destino):
                shutil.move(origen, destino)
```

Inicialmente, movemos todas las imágenes que pueda haber en la carpeta validación a la carpeta entrenamiento, con la finalidad de que cada vez que se entrena, un 20% aleatorio de

cada animal en la carpeta entrenamiento vaya a la validación, cambiando las imágenes y haciendo que el entrenamiento pueda ser más exitoso.

Primero, hacemos un bucle en el que se recorren todas las clases de la carpeta validación y obtenemos la ruta de cada clase (animal), generando la ruta necesaria en la carpeta entrenamiento, ya que, el nombre es el mismo, pero en la otra carpeta.

En caso de que el directorio donde hay que mover las imágenes no exista, se crea.

Después verificamos si de donde estamos cogiendo las imágenes es un directorio o no, con el fin de procesar únicamente los directorios que contengan imágenes.

Por último, si la dirección que contiene la variable “clase_validacion” es un directorio y contiene imágenes, movemos las imágenes de dicho directorio al directorio de destino (el mismo nombre, pero en la carpeta entrenamiento), asegurándonos de que, si la imagen que hay en el directorio, existe en el directorio de destino, no se mueva.

```
# Mover un 20% de las imágenes de entrenamiento a validación
for clase in os.listdir(datos_entrenamiento):
    clase_entrenamiento = os.path.join(datos_entrenamiento, clase)
    clase_validacion = os.path.join(datos_validacion, clase)

    if not os.path.exists(clase_validacion):
        os.makedirs(clase_validacion)

    if os.path.isdir(clase_entrenamiento):
        imagenes_entrenamiento = os.listdir(clase_entrenamiento)
        total_imagenes = len(imagenes_entrenamiento)

        if total_imagenes > 0:
            cantidad_a_mover = int(0.2 * total_imagenes)
            imagenes_a_mover = random.sample(imagenes_entrenamiento, cantidad_a_mover)

            for imagen in imagenes_a_mover:
                origen = os.path.join(clase_entrenamiento, imagen)
                destino = os.path.join(clase_validacion, imagen)

                if not os.path.exists(destino):
                    shutil.move(origen, destino)

print("Proceso de mover un 20% de las imágenes a la carpeta de validación completado.")
```

En la siguiente función, lo que haremos será mover un 20% de las imágenes de cada clase (subdirectorio) del directorio entrenamiento (una vez que tenemos la totalidad de las imágenes en este), al directorio validación, con la finalidad de que las imágenes que se encuentren en validación no siempre sean las mismas, explicado anteriormente.

Recorremos las clases (subdirectorios) de la carpeta entrenamiento y almacenamos en variables la ruta del directorio en la carpeta de entrenamiento y en la carpeta de validación (carpeta de origen y destino).

Nos aseguramos de que la carpeta de destino exista, si no la creamos.

Si la carpeta en la que nos encontramos es un directorio, obtenemos todas las imágenes y obtenemos el número total de las mismas, para comprobar si este número es mayor que 0 (es decir, si hay imágenes en el directorio). Si hay imágenes, calculamos el 20% del número total de imágenes y escogemos de manera aleatoria, ese número imágenes.

Por último, movemos las imágenes seleccionadas, del directorio de origen (de la carpeta entrenamiento), al directorio destino correspondiente (de la carpeta validación) y mostramos un mensaje de validación.

```
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=0)

# Parámetros del modelo
epocas = 150  # Número de épocas
altura = 128  # Altura de las imágenes de entrada
longitud = 128  # Longitud de las imágenes de entrada
batch_size = 32  # Número de imágenes procesadas por lote
pasos = 9500 // batch_size  # Número de pasos por época
pasos_validacion = 2500 // batch_size  # Número de pasos para validación
clases = 19  # Número de clases (una para cada clase de animal)
lr = 0.0001  # Tasa de aprendizaje
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)

directorio = datos_entrenamiento

def mostrar_imagenes_aleatorias(directorio, altura=128, longitud=128, cantidad=5):
    # Obtener la lista de archivos en el directorio
    imagenes = [f for f in os.listdir(directorio) if f.lower().endswith(('png', 'jpg', 'jpeg'))]

    if len(imagenes) < cantidad:
        print("No hay suficientes imágenes en el directorio.")
        return

    imagenes_seleccionadas = random.sample(imagenes, cantidad)

    plt.figure(figsize=(15, 5))
    for i, img_nombre in enumerate(imagenes_seleccionadas):
        img_path = os.path.join(directorio, img_nombre)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (longitud, altura))

        plt.subplot(1, cantidad, i + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(img_nombre)

    plt.show()
```

En este apartado, empezamos a configurar los distintos parámetros que van a ayudar a mejorar el entrenamiento del modelo:

Empezamos por el “early_stop”, que se encarga de parar el entrenamiento, si los valores de precisión y de error no mejoran a lo largo del mismo, para evitar un sobreajuste innecesario del modelo o carga de información inútil para nuestra finalidad. Está configurado para que, en caso de que pasen 10 épocas del entrenamiento sin mejorar, el entrenamiento parará automáticamente y se guardará la época con mejores valores de precisión y error.

Después configuramos las épocas del entrenamiento, las dimensiones de las imágenes, número de imágenes procesadas por lote, el número de pasos por época de entrenamiento y validación, la tasa de aprendizaje, entre otras…

Cabe destacar la configuración de la variable “lr_scheduler”, que va a variar el valor de la tasa de aprendizaje en función de las necesidades del modelo. Cuando el modelo no mejora, la tasa de aprendizaje se verá reducida automáticamente, con la finalidad de que el entrenamiento siga mejorando y el modelo esté más tiempo y se fije más detalladamente en cada imagen. Esto, hará el entrenamiento más lento, pero seguirá mejorando los valores. En este caso, le decimos que se fije en el valor de pérdida de la validación, que la nueva tasa de entrenamiento sea la actual, por 0.2, cuando lleve 5 épocas sin mejorar los valores de entrenamiento del modelo, haciendo que se muestre un mensaje de información cada vez que el valor se reduzca.

Después, tenemos la función de mostrar imágenes aleatorias, en la que se muestran 5 imágenes de cualquier clase de la carpeta entrenamiento, con la dimensión proporcionada a la hora de crear la función.

Para ello, obtenemos todos los archivos de las distintas clases que cumplan con los formatos y hacemos que, si el número de imágenes obtenidas es menor que 5, nos muestre un mensaje de error y que nos alerte de que el número de imágenes en el set es muy escaso, y por lo que, algo va mal.

Si el número de imágenes obtenidas es mayor, seleccionamos de todas esas imágenes, 5 imágenes aleatorias.

Después de realizar la selección, hacemos un bucle en el que se recorran las imágenes seleccionadas y le daremos formato para mostrarlas al usuario y ver si las imágenes se van a ver bien y van a poder ser estudiadas de una manera más fácil por el modelo, o, por el contrario, se van a tener que cambiar de formato a uno más claro. Por último, se muestran las imágenes.

```
# Transformación de las imágenes para entrenamiento (Data Augmentation)
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.3,
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.2,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=20.0,
    vertical_flip=True,
    fill_mode="nearest",
)

# Transformación de las imágenes para validación
validacion_datagen = ImageDataGenerator(
    rescale=1./255,  # Normaliza las imágenes para validación
)
```

Una vez que las imágenes son correctas, pasamos a intentar crear la mayor cantidad de variabilidad de imágenes posible, con distintos efectos y posiciones, para intentar que, el modelo, pueda capturar y reconocer la mayor cantidad de animales posible en la mayor cantidad de situaciones distintas posibles.

Para ello, uso lo que es “Data Augmentation”, que consiste en aumentar la cantidad de datos, haciendo cambios en las fotos, como puede ser zoom, destellos, rotaciones, distorsiones…

Estos cambios les aplicamos en la carpeta de entrenamiento únicamente, manteniendo las imágenes normales en la carpeta de validación y complicando el entrenamiento al modelo, y haciendo que compruebe el entrenamiento (en la carpeta de validación), con imágenes normales, para así no confundir al modelo y únicamente, que tenga presente, esa variabilidad en las imágenes.

Cabe destacar, y me parece importante, indagar en la explicación de la normalización de las imágenes a la hora de entrenar el modelo.

La normalización consiste en cambiar los valores de los pixeles de las imágenes para mejorar la eficiencia del entrenamiento del modelo. Los pixeles de las imágenes van de valores de 0 a 255, lo que hace que los valores sean demasiado grandes para ser captados por un modelo de IA y complicando el entrenamiento de este. Para evitar esto, normalizamos las imágenes, es decir, cambiamos los valores de pixeles actuales a valores de 0 a 1, haciendo que estos estén en un rango más pequeño y consistente, haciendo más pequeña la variabilidad y haciendo más consistente y sencillo el modelo.

```
# Carga de las imágenes de entrenamiento
imagen_entrenamiento = entrenamiento_datagen.flow_from_directory(
    datos_entrenamiento,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical',
)

# Carga de las imágenes de validación
imagen_validacion = validacion_datagen.flow_from_directory(
    datos_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical',
)
```

Procedemos a cargar las imágenes de entrenamiento y validacion, generando los lotes con las imágenes con el “Data Augmentation”, poniendo las dimensiones correctas a la imagen, con el número correcto de imágenes por lote y las etiquetamos de manera categórica (según la clase a la que pertenezcan), haciendo más facil la resolucion del problema de clasificacion múltiple (con mas de 2 clases).

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

# Congelar las capas base del modelo
base_model.trainable = False
```

Cargamos un modelo preentrenado (VGG16), muy conocida por su rendimiento en tareas de clasificación de imágenes. Fue entrenado con el conjunto de imágenes “imagenet”, y nos puede servir como ayuda a la hora de facilitar el reconocimiento de estructuras y patrones en las imágenes del modelo a entrenar. Hacemos que no se usen las capas superiores de clasificación de imágenes final, para hacer que solo se use la parte de captación de características generales de las imágenes, y especificamos la forma de entrada que debe tener la imagen, con las dimensiones y los 3 canales de la imagen (RGB).

Congelamos las capas de este modelo para no entrenarle, evitando cambiar su eficacia y su entrenamiento, haciendo que solo sea útil para visualizar características del nuevo entrenamiento, y este, no afecte a las características particulares del modelo preentrenado.

```python
# Crear el modelo con transfer learning (añadiendo nuevas capas)
cnn = Sequential([
    base_model,  # Modelo base preentrenado
    Flatten(),  # Aplanar la salida de las capas convolucionales
    Dense(1024, activation='relu', kernel_regularizer=l2(0.01)),  # Regularización L2
    Dropout(0.5),  # Aumenta si es necesario
    Dense(512, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(19, activation='softmax')
])


# Compilar el modelo
cnn.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=lr), metrics=['accuracy'])

checkpoint_callback = ModelCheckpoint(
    'modelo_checkpoint_ave.keras',  # Nombre del archivo donde se guarda el modelo
    save_weights_only=False,  # Guarda tanto la estructura como los pesos
    save_freq=30 * pasos,  # Guarda cada 30 épocas (según los pasos por época)
    verbose=1  # Muestra un mensaje cuando se guarda el modelo
)

# Entrenar el modelo
cnn.fit(imagen_entrenamiento, steps_per_epoch=pasos, epochs=epocas, validation_data=imagen_validacion, validation_steps=pasos_validacion, callbacks=

cnn.save('modelo.keras')  # Guarda todo el modelo (estructura + pesos)
```

Seguimos dándole forma al entrenamiento del modelo, usando “TransferLearning”. Definimos un modelo secuencial, es decir, las capas van a ir una detrás de otra (el final de una capa es el comienzo de la siguiente), utilizando:

- El modelo preentrenado antes configurado
- Convertimos las capas convolucionales en capas unidimensionales (necesario para que las salidas puedan ser alimentadas a las capas densas)

A partir de aquí, definimos las capas:

- Dense:
    - Capa de entrada con 1024 neuronas, con la función de activación RELU
    - Regularización L2 (penaliza los pesos grandes, favoreciendo los modelos más simples) con factor de penalización de 0.01, evitando el sobreajuste.
    - Desactivamos mediante “DropOut(0.5)” el 50% de las neuronas aleatoriamente para prevenir el sobreajuste.

    Repetimos estas capas otras 2 veces más con distintas neuronas (512 y 256, respectivamente) con la regularización L2 para evitar el sobreajuste.

- Capa de salida con 19 neuronas, ya que clasificamos 19 clases de animales distintas, con la activación “Softmax”, que permite la clasificación multiclase.

Después, compilamos el modelo (preparamos el modelo para ejecutarlo), utilizando la función de perdida (loss) para problemas de clasificación multiclase, como el nuestro.

Utilizamos el optimizador Adam, ajustando los pesos del entrenamiento y utilizando el “learning rate” definido previamente.

Medimos la precisión del modelo mediante la variable “accuracy”.

A la hora de compilar, lo que hacemos es decir al modelo cómo aprender, qué método usar para mejorar en cada iteración y cómo medir el rendimiento.

Justo debajo, configuramos un “checkpoint”, es decir, un punto en donde los pesos del modelo se van almacenando de manera automática en un archivo aparte. Guardamos tanto la

estructura como los pesos del modelo, con una frecuencia de 30 épocas y mostramos un mensaje cada vez que se guarde, para saber que se está realizando correctamente.

Entrenamos el modelo con los valores configurados (épocas, pasos, imágenes, callbacks…) y cuando acabe el entrenamiento, guardaremos los valores finales en el archivo indicado en la última línea.

Siguiendo este modelo de configuración de entrenamiento y capas, hacemos un modelo más eficiente, evitando el sobreajuste del modelo y por lo tanto creando un modelo más genérico, captando más información al principio del entrenamiento y haciendo que esta información se vaya filtrando cada vez más, quedando en el modelo, unas características más claras y eficientes a la hora de reconocer dichas imágenes y formando una especie de “embudo”, que hace que la información se vaya colando con un número cada vez menor de neuronas y haciendo un mejor desarrollo del modelo.

Otro punto a tener en cuenta es el por qué usamos una neurona en la capa de salida por cada clase, y es que necesitamos que el modelo devuelva una probabilidad para cada clase, dando un valor del 0 al 1. Al tener 19 clases, generamos una neurona especializada en cada clase, con un valor de probabilidad. El modelo escogerá el modelo con una tasa probabilidad mayor con la finalidad de tener un mayor acierto a la hora de reconocerlo.

Este modelo tenía una complejidad bastante alta, teniendo en cuenta que, la cantidad de imágenes del set era baja (unas 12000 imágenes entre la carpeta de validación y la carpeta de entrenamiento) y la cantidad de animales distintos (clases), era relativamente alta (19), teniendo animales muy semejantes entre sí y haciendo que la IA diese unos resultados insuficientes, siendo de una precisión del 52%, lo que provocaba muchos errores en el reconocimiento de estas imágenes. Además, hay que saber, que la IA en el reconocimiento de imágenes, hace que llegar a unos valores de precisión buenos (entre el 60% y el 80%) sea bastante más complicado.

DEFINICIÓN NUEVOS REQUISITOS EN IA DE RECONOCIMIENTO ANIMAL

Tras la realización de este modelo, se sugirieron unos cambios en los requisitos del proyecto, siendo los siguientes.

El modelo ya no tenía que reconocer ciertos animales salvajes, sino, que el proyecto consistía en lo siguiente.

- Reconocer:

ANIMALES DOMÉSTICOS O DE GANADO EN CASTILLA Y LEÓN

Oveja, cabra, toro, vaca, perro, gato, cerdo, gallina, pollo, bisonte, caballo, burro, pavo, hurón, cobaya, hámster, avejas, cisne, caracoles, codorniz, gallinas

ANIMALES SALVAJES MAMÍFEROS TERRESTRES EN CASTILLA Y LEÓN

Lince ibérico, lobo, ciervo, jabalí, gamo, corzo, zorro, tejón, muflón, nutria, erizo, marta, lirón, murciélago, ardilla, liebre, marmota, gato montés, topillo, tejón, oso pardo

ANIMALES SALVAJES REPTILES Y ANFIBIOS TERRESTRES EN CASTILLA Y LEÓN

Culebra, víbora, tortuga, lagarto, lagartija, salamanquesa, salamandra, sapo corredor, rana común, tritón

AVES SALVAJES EN CASTILLA Y LEÓN

Buitre leonado, águila real, águila imperial, cernícalo común, halcón peregrino, lechuza, alimoche, cigüeña, grulla, búho, paloma, ganso,

INSECTOS EN CASTILLA Y LEÓN

Mariposa, escarabajo, grillo común, amantis religiosa, mosca común

ARÁCNIDOS EN CASTILLA Y LEÓN

Araña, escorpión, tarántula, ácaros, garrapatas, opiliones, solífugos.

Algunos de estos animales, han tenido que ser descartados desde el principio, ya sea por la similitud que existen entre ellos, el tamaño de los animales (que sea difícil reconocerlos a 50 metros de altura) o, en algunos casos, por la dificultad de acceder a set de imágenes completos para que el modelo pueda reconocerlos con el entrenamiento. Estas excepciones, las mencionaremos más adelante.

- Modelar una IA, capaz de reconocer mediante imágenes los distintos animales del territorio autonómico, mediante cámaras implantadas en un dron, que sobrevolaría a unos 50 metros de altura, teniendo que distinguir:
    - Animales domésticos o de ganado existentes en Castilla y León
    - Animales salvajes terrestres existentes en bosques y campos de Castilla y León
    - Reptiles y anfibios existentes en el medio autonómico
    - Aves (siempre que no estén en vuelo), existentes en Castilla y León

  Para conseguir solventar de una mejor manera estos cambios propuestos, he llegado a diversas conclusiones que harían que el modelo y el proyecto tuviesen un mejor resultado final:

1. **Implantación de una cámara térmica en el dron**

Es muy útil la implantación de una cámara térmica en el dron, junto a la ya existente RGB, para detectar de manera fácil la existencia de un animal en la imagen, con la excepción de ciertos reptiles y anfibios como ciertas serpientes, que no serán detectadas con facilidad en la cámara térmica.

2. **Implantación de una IA para recortar las imágenes**

La cámara térmica, lo que nos va a permitir es que, podamos crear una IA, que con la obtención de una imagen térmica y otra imagen RGB, pueda detectar el punto de calor de la imagen térmica, es decir, el animal, haciendo que se recorte la imagen RGB, dejando únicamente el animal en la imagen y detectándolo con más facilidad, además de evitar muchos errores con la confusión de algún animal con el medio.

3. **Implantación de distintas IA para los distintos tipos de animales**

Anteriormente, hemos distinguido los animales a reconocer en 4 grupos distintos debido a las distintas características que tienen en común:

1. Tipo de animal
2. Medio en el que habita (parcela o salvaje)

De esta forma, podremos crear modelos más efectivos (con menor número de clases), que pueda distinguir mejor cada animal.

Además, de esta forma, podremos jugar con la activación de unos modelos u otros en función de las necesidades en cada momento. Por ejemplo:

- Cuando el dron sobrevuele una parcela privada, activaremos únicamente el modelo que reconoce los animales domésticos o de ganado, junto con el térmico.
- Cuando el dron sobrevuele un medio salvaje, activaremos todos los modelos, junto con el térmico, para evitar que se nos pase algo.

Así, podremos estructurar de mejor manera el funcionamiento de los distintos modelos, haciéndolo con más efectivo y con menos cantidad de errores.

Por último, puntualizaremos el orden de ejecución de los distintos modelos, ya que cada uno, dota de características distintas.

En caso de sobrevolar una parcela privada:

Activaremos primero el modelo de reptiles y anfibios (por si alguno de ellos se cuela en estas parcelas). Después el modelo térmico, y, por último, el modelo doméstico o de ganado.

En caso de sobrevolar un entorno natural salvaje:

Activaremos primero el modelo de reptiles y anfibios. Después el modelo térmico. Posteriormente el modelo reptiles y anfibios de nuevo, seguido del modelo de aves, y, por último, el modelo de reconocimiento de animales salvajes.

Este seguimiento está sujeto a cambios según se vaya creando el proyecto, con la finalidad de un mejor funcionamiento del proyecto.

De esta forma, podremos cubrir la mayor parte de excepciones posible y captar la mayor cantidad de animales, dotando al proyecto de una mayor eficacia y un menor porcentaje de error.

MODELOS DE IA DEFINITIVOS DE RECONOCIMIENTO ANIMAL

Para la realización de estos modelos, he usado las mismas herramientas que con el modelo anterior, el lenguaje de programación Python, la herramienta gráfica Jupyter y la misma estructuración del código, ya que, con una reducción de clases por modelo, es posible que el modelo mejore por su simplificación.

DESARROLLO MODELOS ANIMALES DOMÉSTICOS

Empecemos por el modelo creado para el reconocimiento de animales domésticos o de ganado:

Los animales para los que el modelo está entrenado a reconocer son:

Oveja, cabra, toro, vaca, perro, gato, cerdo, gallina, bisonte, caballo, burro, pavo, cisne y conejo.

(Los animales que no están en la lista, han sido eliminados por cuestiones mencionadas antes, como longitud o similitud entre ellos)

Siendo el toro el único que el modelo no está entrenado para reconocer, debido a que no ha sido posible encontrar sets de imágenes relacionado con ese animal. A partir de ahí, está pensado introducir el animal posteriormente cuando sea posible acceder a la cantidad de imágenes demandada para que este pueda ser reconocido.

Con estos animales intentamos adaptarnos a todo tipo de fincas.

```
import os
import cv2
import random
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras import backend as K
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.applications import VGG16
import shutil
from tensorflow.keras.regularizers import l2
```

```
# Limpiar la sesión de Keras
K.clear_session()

# Directorios de imágenes
datos_entrenamiento = '../Domestico/AnimalesDomesticos'  # Cambiar a la ruta correcta
datos_validacion = '../Domestico/AnimalesValidacionDomesticos'  # Cambiar a la ruta correcta

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="./logs", histogram_freq=1)
```

Cambiamos la dirección de donde debe de coger las imágenes para entrenar el nuevo modelo.

```python
# Función para redimensionar sin distorsión, manteniendo la relación de aspecto
def redimensionar_manteniendo_relacion_aspecto(img, tamaño_max=224):
    altura, ancho = img.shape[:2]
    factor_escala = tamaño_max / float(max(altura, ancho))

    nuevo_ancho = int(ancho * factor_escala)
    nueva_altura = int(altura * factor_escala)

    # Redimensionar la imagen con una interpolación de alta calidad
    img_redimensionada = cv2.resize(img, (nuevo_ancho, nueva_altura), interpolation=cv2.INTER_AREA)

    # Crear una imagen de fondo cuadrada (100x100) con color negro
    fondo = np.zeros((tamaño_max, tamaño_max, 3), dtype=np.uint8)

    # Calcular las coordenadas para centrar la imagen en el fondo
    x_offset = (tamaño_max - nuevo_ancho) // 2
    y_offset = (tamaño_max - nueva_altura) // 2

    # Copiar la imagen redimensionada en el centro del fondo negro
    fondo[y_offset:y_offset + nueva_altura, x_offset:x_offset + nuevo_ancho] = img_redimensionada

    return fondo
```

```python
# Mover todas las imágenes de validación a entrenamiento al inicio
for clase in os.listdir(datos_validacion):
    clase_validacion = os.path.join(datos_validacion, clase)
    clase_entrenamiento = os.path.join(datos_entrenamiento, clase)

    if not os.path.exists(clase_entrenamiento):
        os.makedirs(clase_entrenamiento)

    if os.path.isdir(clase_validacion):
        imagenes = os.listdir(clase_validacion)
        for imagen in imagenes:
            origen = os.path.join(clase_validacion, imagen)
            destino = os.path.join(clase_entrenamiento, imagen)

            if not os.path.exists(destino):
                shutil.move(origen, destino)
```

```python
# Mover un 20% de las imágenes de entrenamiento a validación
for clase in os.listdir(datos_entrenamiento):
    clase_entrenamiento = os.path.join(datos_entrenamiento, clase)
    clase_validacion = os.path.join(datos_validacion, clase)

    # Asegurarse de que exista la carpeta de validación para la clase
    if not os.path.exists(clase_validacion):
        os.makedirs(clase_validacion)

    if os.path.isdir(clase_entrenamiento):
        imagenes_entrenamiento = os.listdir(clase_entrenamiento)
        total_imagenes = len(imagenes_entrenamiento)

        if total_imagenes > 0:
            cantidad_a_mover = int(0.2 * total_imagenes)
            imagenes_a_mover = random.sample(imagenes_entrenamiento, cantidad_a_mover)

            for imagen in imagenes_a_mover:
                origen = os.path.join(clase_entrenamiento, imagen)
                destino = os.path.join(clase_validacion, imagen)

                if not os.path.exists(destino):
                    shutil.move(origen, destino)

print("Proceso de mover un 20% de las imágenes a la carpeta de validación completado.")
```

```
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=0)

# Parámetros del modelo
epocas = 120  # Número de épocas
altura = 128  # Altura de las imágenes de entrada
longitud = 128  # Longitud de las imágenes de entrada
batch_size = 32  # Número de imágenes procesadas por lote
pasos = 6000 // batch_size  # Número de pasos por época
pasos_validacion = 1800 // batch_size  # Número de pasos para validación
clases = 14  # Número de clases (una para cada clase de animal)
lr = 0.0001  # Tasa de aprendizaje
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)

directorio = datos_entrenamiento

def mostrar_imagenes_aleatorias(directorio, altura=128, longitud=128, cantidad=5):
    # Obtener la lista de archivos en el directorio
    imagenes = [f for f in os.listdir(directorio) if f.lower().endswith(('png', 'jpg', 'jpeg'))]

    if len(imagenes) < cantidad:
        print("No hay suficientes imágenes en el directorio.")
        return

    imagenes_seleccionadas = random.sample(imagenes, cantidad)

    plt.figure(figsize=(15, 5))
    for i, img_nombre in enumerate(imagenes_seleccionadas):
        img_path = os.path.join(directorio, img_nombre)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (longitud, altura))

        plt.subplot(1, cantidad, i + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(img_nombre)

    plt.show()
```

He cambiado la cantidad de épocas a la que he creído más adaptada para la cantidad de información del nuevo modelo, así como el número de pasos de validación y de pasos por época.

```
# Transformación de las imágenes para entrenamiento (Data Augmentation)
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.3,
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.2,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=20.0,
    vertical_flip=True,
    fill_mode="nearest",
)

# Transformación de las imágenes para validación
validacion_datagen = ImageDataGenerator(
    rescale=1./255,  # Normaliza las imágenes para validación
)
```

```
# Carga de las imágenes de entrenamiento
imagen_entrenamiento = entrenamiento_datagen.flow_from_directory(
    datos_entrenamiento,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical',
)

# Carga de las imágenes de validación
imagen_validacion = validacion_datagen.flow_from_directory(
    datos_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical',
)
```

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

# Congelar las capas base del modelo
base_model.trainable = False
```

```
# Crear el modelo con transfer learning (añadiendo nuevas capas)
cnn = Sequential([
    base_model,  # Modelo base preentrenado
    Flatten(),  # Aplanar la salida de las capas convolucionales
    Dense(1024, activation='relu', kernel_regularizer=l2(0.01)),  # Regularización L2
    Dropout(0.5),  # Aumenta si es necesario
    Dense(512, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(14, activation='softmax')  # Cambio a 19 clases
])

# Compilar el modelo
cnn.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=lr), metrics=['accuracy'])

checkpoint_callback = ModelCheckpoint(
    'modelo_checkpoint.keras',  # Nombre del archivo donde se guarda el modelo
    save_weights_only=False,  # Guarda tanto la estructura como los pesos
    save_freq=15 * pasos,  # Guarda cada 15 épocas (según los pasos por época)
    verbose=1  # Muestra un mensaje cuando se guarda el modelo
)

# Entrenar el modelo
cnn.fit(imagen_entrenamiento, steps_per_epoch=pasos, epochs=epocas, validation_data=imagen_validacion, validation_steps=pasos_validacion, callbacks=[early_stop, lr_scheduler, tensorboard_callback, checkpoint_callback])

cnn.save('modeloDomestico.keras')  # Guarda todo el modelo (estructura + pesos)
```

El resultado del primer entrenamiento de los modelos fue mejor que el resultado del modelo anterior, aunque no es lo que buscamos del todo. Estos fueron los resultados:

```
Epoch 120/120
187/187 ━━━━━━━━━━━━━━━━━━━━ 88s 463ms/step - accuracy: 0.4694 - loss: 1.9827 - val_accuracy: 0.5866 - val_loss:
1.5844 - learning_rate: 4.0000e-06
```

Ha sido una duración de 120 épocas, en la que la última tuvo estos resultados:

Una precisión del 46% y un valor de error de 1.98 en las imágenes de entrenamiento y una precisión del 58% y del 1.58 del valor de error en las imágenes de validación.

No es un mal resultado, pero cambiando algún valor de entrenamiento como el batch_size, el número de épocas (ampliándolo para ampliar el margen de mejora del modelo) y algunos de los valores del DataAugmentation, reduciéndolos para no confundir al modelo y poder tener unos mejores valores.

DESARROLLO MODELO DE AVES SALVAJES

Seguimos con el modelo para el reconocimiento de aves salvajes en el territorio. Los animales que van a ser configurados para que el modelo reconozca serán los siguientes (por parte de las aves, por tema de tamaño y similitud entre ellas):
Buitre leonado, águila real, halcón peregrino, lechuza, alimoche, cigüeña, grulla y gansos.

(Los animales que no están en la lista, han sido eliminados por cuestiones mencionadas antes, como longitud o similitud entre ellos).

Siendo el alimoche el animal que no está presentado en el set de datos conjunto para entrenar este modelo, por la misma causa que el toro en el anterior (no hay sets de imágenes lo

suficientemente grandes como para que sea reconocido), por lo que se añadirá cuando se puedan acceder a sets con la longitud suficiente para poder llevarse a cabo.

```
import os
import cv2
import random
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras import backend as K
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.applications import VGG16
import shutil
from tensorflow.keras.regularizers import l2
```

```
# Limpiar la sesión de Keras
K.clear_session()

# Directorios de imágenes
datos_entrenamiento = '../Aves/Entrenamiento'
datos_validacion = '../Aves/Validacion'

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="./logs", histogram_freq=1)
```

Cambiamos el directorio de donde serán recogidas las imágenes.

```
# Función para redimensionar sin distorsión, manteniendo la relación de aspecto
def redimensionar_manteniendo_relacion_aspecto(img, tamaño_max=224):
    altura, ancho = img.shape[:2]
    factor_escala = tamaño_max / float(max(altura, ancho))

    nuevo_ancho = int(ancho * factor_escala)
    nueva_altura = int(altura * factor_escala)

    # Redimensionar la imagen con una interpolación de alta calidad
    img_redimensionada = cv2.resize(img, (nuevo_ancho, nueva_altura), interpolation=cv2.INTER_AREA)

    # Crear una imagen de fondo cuadrada (100x100) con color negro
    fondo = np.zeros((tamaño_max, tamaño_max, 3), dtype=np.uint8)

    # Calcular las coordenadas para centrar la imagen en el fondo
    x_offset = (tamaño_max - nuevo_ancho) // 2
    y_offset = (tamaño_max - nueva_altura) // 2

    # Copiar la imagen redimensionada en el centro del fondo negro
    fondo[y_offset:y_offset + nueva_altura, x_offset:x_offset + nuevo_ancho] = img_redimensionada

    return fondo
```

```python
# Mover todas las imágenes de validación a entrenamiento al inicio
for clase in os.listdir(datos_validacion):
    clase_validacion = os.path.join(datos_validacion, clase)
    clase_entrenamiento = os.path.join(datos_entrenamiento, clase)

    if not os.path.exists(clase_entrenamiento):
        os.makedirs(clase_entrenamiento)

    if os.path.isdir(clase_validacion):
        imagenes = os.listdir(clase_validacion)
        for imagen in imagenes:
            origen = os.path.join(clase_validacion, imagen)
            destino = os.path.join(clase_entrenamiento, imagen)

            if not os.path.exists(destino):
                shutil.move(origen, destino)
```

```python
# Mover un 20% de las imágenes de entrenamiento a validación
for clase in os.listdir(datos_entrenamiento):
    clase_entrenamiento = os.path.join(datos_entrenamiento, clase)
    clase_validacion = os.path.join(datos_validacion, clase)

    # Asegurarse de que exista la carpeta de validación para la clase
    if not os.path.exists(clase_validacion):
        os.makedirs(clase_validacion)

    if os.path.isdir(clase_entrenamiento):
        imagenes_entrenamiento = os.listdir(clase_entrenamiento)
        total_imagenes = len(imagenes_entrenamiento)

        if total_imagenes > 0:
            cantidad_a_mover = int(0.2 * total_imagenes)
            imagenes_a_mover = random.sample(imagenes_entrenamiento, cantidad_a_mover)

            for imagen in imagenes_a_mover:
                origen = os.path.join(clase_entrenamiento, imagen)
                destino = os.path.join(clase_validacion, imagen)

                if not os.path.exists(destino):
                    shutil.move(origen, destino)

print("Proceso de mover un 20% de las imágenes a la carpeta de validación completado.")
```

```
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=0)

# Parámetros del modelo
epocas = 150  # Número de épocas
altura = 128  # Altura de las imágenes de entrada
longitud = 128  # Longitud de las imágenes de entrada
batch_size = 32  # Número de imágenes procesadas por lote
pasos = 4500 // batch_size  # Número de pasos por época
pasos_validacion = 1100 // batch_size  # Número de pasos para validación
clases = 8  # Número de clases (una para cada clase de animal)
lr = 0.0001  # Tasa de aprendizaje
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)

directorio = datos_entrenamiento

def mostrar_imagenes_aleatorias(directorio, altura=128, longitud=128, cantidad=5):
    # Obtener la lista de archivos en el directorio
    imagenes = [f for f in os.listdir(directorio) if f.lower().endswith(('png', 'jpg', 'jpeg'))]

    if len(imagenes) < cantidad:
        print("No hay suficientes imágenes en el directorio.")
        return

    imagenes_seleccionadas = random.sample(imagenes, cantidad)

    plt.figure(figsize=(15, 5))
    for i, img_nombre in enumerate(imagenes_seleccionadas):
        img_path = os.path.join(directorio, img_nombre)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (longitud, altura))

        plt.subplot(1, cantidad, i + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(img_nombre)

    plt.show()
```

He cambiado la cantidad de épocas a la que he creído más adaptada para la cantidad de información del nuevo modelo, así como el número de pasos de validación y de pasos por época.

```
# Transformación de las imágenes para entrenamiento (Data Augmentation)
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.3,
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.2,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=20.0,
    vertical_flip=True,
    fill_mode="nearest",
)

# Transformación de las imágenes para validación
validacion_datagen = ImageDataGenerator(
    rescale=1./255,  # Normaliza las imágenes para validación
)
```

```python
# Carga de las imágenes de entrenamiento
imagen_entrenamiento = entrenamiento_datagen.flow_from_directory(
    datos_entrenamiento,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical',
)

# Carga de las imágenes de validación
imagen_validacion = validacion_datagen.flow_from_directory(
    datos_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical',
)
```

```python
# Cargar el modelo preentrenado (VGG16)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(altura, longitud, 3))

# Congelar las capas base del modelo
base_model.trainable = False
```

```python
# Crear el modelo con transfer learning (añadiendo nuevas capas)
cnn = Sequential([
    base_model,  # Modelo base preentrenado
    Flatten(),  # Aplanar la salida de las capas convolucionales
    Dense(1024, activation='relu', kernel_regularizer=l2(0.01)),  # Regularización L2
    Dropout(0.5),  # Aumenta si es necesario
    Dense(512, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(8, activation='softmax')
])


# Compilar el modelo
cnn.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=lr), metrics=['accuracy'])

checkpoint_callback = ModelCheckpoint(
    'modelo_checkpoint_ave.keras',  # Nombre del archivo donde se guarda el modelo
    save_weights_only=False,  # Guarda tanto la estructura como los pesos
    save_freq=15 * pasos,  # Guarda cada 15 épocas (según los pasos por época)
    verbose=1  # Muestra un mensaje cuando se guarda el modelo
)

# Entrenar el modelo
cnn.fit(imagen_entrenamiento, steps_per_epoch=pasos, epochs=epocas, validation_data=imagen_validacion, validation_steps=pasos_validacion, callbacks=[early_stop, lr_scheduler, tensorboard_callback, checkpoint_callback])

cnn.save('modeloAve.keras')  # Guarda todo el modelo (estructura + pesos)
```

Una vez entrenado el modelo, los resultados de este han sido los siguientes:

```
Epoch 150/150
  3/140 ━━━━━━━━━━━━━━━━━━━━ 1:52 821ms/step - accuracy: 0.4479 - loss: 1.4390
Epoch 150: ReduceLROnPlateau reducing learning rate to 3.999999898951501e-06.
140/140 ━━━━━━━━━━━━━━━━━━━━ 31s 216ms/step - accuracy: 0.4989 - loss: 1.4063 - val_accuracy: 0.6029 - val_loss: 1.
2667 - learning_rate: 2.0000e-05
```

El entrenamiento ha tenido una duración de 150 épocas y se han obtenido unos buenos resultados, siendo:

Una precisión del casi 50% y un valor de error de 1.40, no siendo del todo malo en las imágenes de entrenamiento.

Por otra parte, en los valores de validación, se han obtenido unos valores del 60% de precisión y de 1.26 de valor de error.

Los resultados del modelo han sido válidos, pero retocaré los valores de batch_size y DataAugmentation para ver si el valor del entrenamiento mejora.

DESARROLLO MODELO DE REPTILES

Seguimos con el modelo de reptiles y anfibios, en el que entrenamos distintos animales para que el modelo pueda reconocerlos. Estos animales son capaces de camuflarse de manera excepcional con el entorno. Para poder distinguirlos, haremos uso de la cámara térmica, aunque no en todas las situaciones.

Las serpientes y otros animales, en muchas ocasiones, no tienen una temperatura corporal caliente, por lo que la cámara térmica no detectaría un animal (un punto de calor) en la imagen. Para solucionarlo, la cámara térmica, no será usada para este modelo de IA.

Algunos de los animales comentados para diferenciar, han sido eliminados por cuestión de tamaño o similitud entre ellos o el entorno, de manera que, estos son los animales por diferenciar para el modelo:

Culebra (serpientes), lagarto, sapo corredor y tortuga.

El modelo, ha sido configurado de igual manera que los demás modelos (misma estructura):

```
import os
import cv2
import random
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras import backend as K
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.applications import VGG16
import shutil
from tensorflow.keras.regularizers import l2
```

```
# Limpiar la sesión de Keras
K.clear_session()

# Directorios de imágenes
datos_entrenamiento = '../Reptiles/Entrenamiento'
datos_validacion = '../Reptiles/Validacion'

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="./logs", histogram_freq=1)
```

```python
# Función para redimensionar sin distorsión, manteniendo la relación de aspecto
def redimensionar_manteniendo_relacion_aspecto(img, tamaño_max=224):
    altura, ancho = img.shape[:2]
    factor_escala = tamaño_max / float(max(altura, ancho))

    nuevo_ancho = int(ancho * factor_escala)
    nueva_altura = int(altura * factor_escala)

    # Redimensionar la imagen con una interpolación de alta calidad
    img_redimensionada = cv2.resize(img, (nuevo_ancho, nueva_altura), interpolation=cv2.INTER_AREA)

    # Crear una imagen de fondo cuadrada (100x100) con color negro
    fondo = np.zeros((tamaño_max, tamaño_max, 3), dtype=np.uint8)

    # Calcular las coordenadas para centrar la imagen en el fondo
    x_offset = (tamaño_max - nuevo_ancho) // 2
    y_offset = (tamaño_max - nueva_altura) // 2

    # Copiar la imagen redimensionada en el centro del fondo negro
    fondo[y_offset:y_offset + nueva_altura, x_offset:x_offset + nuevo_ancho] = img_redimensionada

    return fondo
```



```python
# Mover todas las imágenes de validación a entrenamiento al inicio
for clase in os.listdir(datos_validacion):
    clase_validacion = os.path.join(datos_validacion, clase)
    clase_entrenamiento = os.path.join(datos_entrenamiento, clase)

    if not os.path.exists(clase_entrenamiento):
        os.makedirs(clase_entrenamiento)

    if os.path.isdir(clase_validacion):
        imagenes = os.listdir(clase_validacion)
        for imagen in imagenes:
            origen = os.path.join(clase_validacion, imagen)
            destino = os.path.join(clase_entrenamiento, imagen)

            if not os.path.exists(destino):
                shutil.move(origen, destino)
```

```python
# Mover un 20% de las imágenes de entrenamiento a validación
for clase in os.listdir(datos_entrenamiento):
    clase_entrenamiento = os.path.join(datos_entrenamiento, clase)
    clase_validacion = os.path.join(datos_validacion, clase)

    # Asegurarse de que exista la carpeta de validación para la clase
    if not os.path.exists(clase_validacion):
        os.makedirs(clase_validacion)

    if os.path.isdir(clase_entrenamiento):
        imagenes_entrenamiento = os.listdir(clase_entrenamiento)
        total_imagenes = len(imagenes_entrenamiento)

        if total_imagenes > 0:
            cantidad_a_mover = int(0.2 * total_imagenes)
            imagenes_a_mover = random.sample(imagenes_entrenamiento, cantidad_a_mover)

            for imagen in imagenes_a_mover:
                origen = os.path.join(clase_entrenamiento, imagen)
                destino = os.path.join(clase_validacion, imagen)

                if not os.path.exists(destino):
                    shutil.move(origen, destino)

print("Proceso de mover un 20% de las imágenes a la carpeta de validación completado.")
```

```
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=0)

# Parámetros del modelo
epocas = 120  # Número de épocas
altura = 128  # Altura de las imágenes de entrada
longitud = 128  # Longitud de las imágenes de entrada
batch_size = 32  # Número de imágenes procesadas por lote
pasos = 6000 // batch_size  # Número de pasos por época
pasos_validacion = 1800 // batch_size  # Número de pasos para validación
clases = 5  # Número de clases (una para cada clase de animal)
lr = 0.0001  # Tasa de aprendizaje
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)

directorio = datos_entrenamiento

def mostrar_imagenes_aleatorias(directorio, altura=128, longitud=128, cantidad=5):
    # Obtener la lista de archivos en el directorio
    imagenes = [f for f in os.listdir(directorio) if f.lower().endswith(('png', 'jpg', 'jpeg'))]

    if len(imagenes) < cantidad:
        print("No hay suficientes imágenes en el directorio.")
        return

    imagenes_seleccionadas = random.sample(imagenes, cantidad)

    plt.figure(figsize=(15, 5))
    for i, img_nombre in enumerate(imagenes_seleccionadas):
        img_path = os.path.join(directorio, img_nombre)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (longitud, altura))

        plt.subplot(1, cantidad, i + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(img_nombre)

    plt.show()
```

```
# Transformación de las imágenes para entrenamiento (Data Augmentation)
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.3,
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.2,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=20.0,
    vertical_flip=True,
    fill_mode="nearest",
)

# Transformación de las imágenes para validación
validacion_datagen = ImageDataGenerator(
    rescale=1./255,  # Normaliza las imágenes para validación
)
```

```
# Carga de las imágenes de entrenamiento
imagen_entrenamiento = entrenamiento_datagen.flow_from_directory(
    datos_entrenamiento,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical',
)

# Carga de las imágenes de validación
imagen_validacion = validacion_datagen.flow_from_directory(
    datos_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical',
)
```

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

# Congelar las capas base del modelo
base_model.trainable = False
```

```
# Crear el modelo con transfer learning (añadiendo nuevas capas)
cnn = Sequential([
    base_model,  # Modelo base preentrenado
    Flatten(),  # Aplanar la salida de las capas convolucionales
    Dense(1024, activation='relu', kernel_regularizer=l2(0.01)),  # Regularización L2
    Dropout(0.5),
    Dense(512, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(5, activation='softmax')
])


# Compilar el modelo
cnn.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=lr), metrics=['accuracy'])

checkpoint_callback = ModelCheckpoint(
    'modelo_checkpoint.keras',  # Nombre del archivo donde se guarda el modelo
    save_weights_only=False,  # Guarda tanto la estructura como los pesos
    save_freq=15 * pasos,  # Guarda cada 15 épocas (según los pasos por época)
    verbose=1  # Muestra un mensaje cuando se guarda el modelo
)

# Entrenar el modelo
cnn.fit(imagen_entrenamiento, steps_per_epoch=pasos, epochs=epocas, validation_data=imagen_validacion, validation_steps=pasos_validacion, callbacks=[early_stop, lr_scheduler,

cnn.save('modeloReptiles.keras')  # Guarda todo el modelo (estructura + pesos)
```


Una vez entrenado el modelo, los resultados de este han sido buenos. Aquí, no vamos a cambiar ningún parámetro para mejorar el modelo, ya que, la precisión de este es correcta. Estos han sido los valores:

```
187/187 ━━━━━━━━━━━━━━━━━━━━ 199s 1s/step - accuracy: 0.6831 - loss: 0.9652 - val_accuracy: 0.7161 - val_loss: 0.87
64 - learning_rate: 8.0000e-07
```

Una precisión del 68% y de menos de 1 en los valores de error (0.96) en la imagen de entrenamiento.

Una precisión del 71% y un valor de error (0.87), siendo valores de error bajos y precisos para ello.

DESARROLLO DE ANIMALES SALVAJES

Por último, hemos configurado el valor de los modelos de los animales salvajes. Los animales configurados para que reconozca el modelo son:

Ardilla, ciervo, jabalí, liebre, lince, lobo, marmota, nutria, oso, tejón y zorro.

Los animales eliminados han sido eliminados debido a las malas condiciones o el tamaño para reconocer este tipo de animales.

La configuración del modelo para los animales salvajes es la misma que los demás, con la misma estructura y valores:

```
import os
import cv2
import random
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras import backend as K
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.applications import VGG16
import shutil
from tensorflow.keras.regularizers import l2
```

```
# Limpiar la sesión de Keras
K.clear_session()

# Directorios de imágenes
datos_entrenamiento = '../Salvaje/Entrenamiento'
datos_validacion = '../Salvaje/Validacion'

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="./logs", histogram_freq=1)
```

```
# Función para redimensionar sin distorsión, manteniendo la relación de aspecto
def redimensionar_manteniendo_relacion_aspecto(img, tamaño_max=224):
    altura, ancho = img.shape[:2]
    factor_escala = tamaño_max / float(max(altura, ancho))

    nuevo_ancho = int(ancho * factor_escala)
    nueva_altura = int(altura * factor_escala)

    # Redimensionar la imagen con una interpolación de alta calidad
    img_redimensionada = cv2.resize(img, (nuevo_ancho, nueva_altura), interpolation=cv2.INTER_AREA)

    # Crear una imagen de fondo cuadrada (100x100) con color negro
    fondo = np.zeros((tamaño_max, tamaño_max, 3), dtype=np.uint8)

    # Calcular las coordenadas para centrar la imagen en el fondo
    x_offset = (tamaño_max - nuevo_ancho) // 2
    y_offset = (tamaño_max - nueva_altura) // 2

    # Copiar la imagen redimensionada en el centro del fondo negro
    fondo[y_offset:y_offset + nueva_altura, x_offset:x_offset + nuevo_ancho] = img_redimensionada

    return fondo
```

```
# Mover todas las imágenes de validación a entrenamiento al inicio
for clase in os.listdir(datos_validacion):
    clase_validacion = os.path.join(datos_validacion, clase)
    clase_entrenamiento = os.path.join(datos_entrenamiento, clase)

    if not os.path.exists(clase_entrenamiento):
        os.makedirs(clase_entrenamiento)

    if os.path.isdir(clase_validacion):
        imagenes = os.listdir(clase_validacion)
        for imagen in imagenes:
            origen = os.path.join(clase_validacion, imagen)
            destino = os.path.join(clase_entrenamiento, imagen)

            if not os.path.exists(destino):
                shutil.move(origen, destino)
```

```python
# Mover un 20% de las imágenes de entrenamiento a validación
for clase in os.listdir(datos_entrenamiento):
    clase_entrenamiento = os.path.join(datos_entrenamiento, clase)
    clase_validacion = os.path.join(datos_validacion, clase)

    # Asegurarse de que exista la carpeta de validación para la clase
    if not os.path.exists(clase_validacion):
        os.makedirs(clase_validacion)

    if os.path.isdir(clase_entrenamiento):
        imagenes_entrenamiento = os.listdir(clase_entrenamiento)
        total_imagenes = len(imagenes_entrenamiento)

        if total_imagenes > 0:
            cantidad_a_mover = int(0.2 * total_imagenes)
            imagenes_a_mover = random.sample(imagenes_entrenamiento, cantidad_a_mover)

            for imagen in imagenes_a_mover:
                origen = os.path.join(clase_entrenamiento, imagen)
                destino = os.path.join(clase_validacion, imagen)

                if not os.path.exists(destino):
                    shutil.move(origen, destino)

print("Proceso de mover un 20% de las imágenes a la carpeta de validación completado.")
```

```python
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=0)

# Parámetros del modelo
epocas = 150  # Número de épocas
altura = 128  # Altura de las imágenes de entrada
longitud = 128  # Longitud de las imágenes de entrada
batch_size = 32  # Número de imágenes procesadas por lote
pasos = 6000 // batch_size  # Número de pasos por época
pasos_validacion = 1800 // batch_size  # Número de pasos para validación
clases = 12  # Número de clases (una para cada clase de animal)
lr = 0.0001  # Tasa de aprendizaje
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)

directorio = datos_entrenamiento

def mostrar_imagenes_aleatorias(directorio, altura=128, longitud=128, cantidad=5):
    # Obtener la lista de archivos en el directorio
    imagenes = [f for f in os.listdir(directorio) if f.lower().endswith(('png', 'jpg', 'jpeg'))]

    if len(imagenes) < cantidad:
        print("No hay suficientes imágenes en el directorio.")
        return

    imagenes_seleccionadas = random.sample(imagenes, cantidad)

    plt.figure(figsize=(15, 5))
    for i, img_nombre in enumerate(imagenes_seleccionadas):
        img_path = os.path.join(directorio, img_nombre)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (longitud, altura))

        plt.subplot(1, cantidad, i + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(img_nombre)

    plt.show()
```

```python
# Transformación de las imágenes para entrenamiento (Data Augmentation)
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.3,
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.2,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=20.0,
    vertical_flip=True,
    fill_mode="nearest",
)

# Transformación de las imágenes para validación
validacion_datagen = ImageDataGenerator(
    rescale=1./255,  # Normaliza las imágenes para validación
)
```

```
# Carga de las imágenes de entrenamiento
imagen_entrenamiento = entrenamiento_datagen.flow_from_directory(
    datos_entrenamiento,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical',
)

# Carga de las imágenes de validación
imagen_validacion = validacion_datagen.flow_from_directory(
    datos_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical',
)
```

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

# Congelar las capas base del modelo
base_model.trainable = False
```

```
# Crear el modelo con transfer learning (añadiendo nuevas capas)
cnn = Sequential([
    base_model,  # Modelo base preentrenado
    Flatten(),  # Aplanar la salida de las capas convolucionales
    Dense(1024, activation='relu', kernel_regularizer=l2(0.01)),  # Regularización L2
    Dropout(0.5),
    Dense(512, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(12, activation='softmax')
])


# Compilar el modelo
cnn.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=lr), metrics=['accuracy'])

checkpoint_callback = ModelCheckpoint(
    'modelo_checkpoint.keras',  # Nombre del archivo donde se guarda el modelo
    save_weights_only=False,  # Guarda tanto la estructura como los pesos
    save_freq=15 * pasos,  # Guarda cada 15 épocas (según los pasos por época)
    verbose=1  # Muestra un mensaje cuando se guarda el modelo
)

# Entrenar el modelo
cnn.fit(imagen_entrenamiento, steps_per_epoch=pasos, epochs=epocas, validation_data=imagen_validacion, validation_steps=pasos_validacion, callbacks=[early_stop,

cnn.save('modeloDomestico.keras')  # Guarda todo el modelo (estructura + pesos)
```

El resultado del entrenamiento de este modelo ha sido el siguiente:

```
Epoch 150/150
187/187 ━━━━━━━━━━━━━━━━━━━━ 51s 268ms/step - accuracy: 0.3965 - loss: 2.0393 - val_accuracy: 0.5064 - val_loss: 1.
7643 - learning_rate: 1.6000e-07
```

En este entrenamiento se ha obtenido un casi 40% de precisión y un valor de error de 2.03 en las imágenes de entrenamiento, siendo en la validación unos datos de 50% de precisión con un valor de error del 1.76.

Estos datos son pobres para la cantidad de clases que estamos manejando, por lo que, tocaremos algunos parámetros relacionados con el entrenamiento del modelo para intentar obtener una mejor precisión de dicho modelo.

Estos son los primeros entrenamientos realizados por cada uno de los modelos realizados hasta ahora. A partir de aquí, iremos corrigiendo errores que surjan y depurando la precisión de los modelos con la finalidad de conseguir unos buenos valores para cada uno de ellos.

MEJORA DE LOS MODELOS ENTRENADOS Y RESOLUCIÓN DE ERRORES

MODELO DE ANIMALES DOMÉSTICOS

Con los nuevos parámetros puestos en el modelo para los animales domésticos, los nuevos resultados del entrenamiento de dicho modelo habrán mejorado. Empecemos, viendo los cambios realizados en los parámetros de entrenamiento del modelo:

```
# Parámetros del modelo
epocas = 120  # Número de épocas
altura = 128  # Altura de las imágenes de entrada
longitud = 128  # Longitud de las imágenes de entrada
batch_size = 32  # Número de imágenes procesadas por lote
pasos = 6000 // batch_size  # Número de pasos por época
pasos_validacion = 1800 // batch_size  # Número de pasos para validación
clases = 14  # Número de clases (una para cada clase de animal)
lr = 0.0001  # Tasa de aprendizaje
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)
```

```
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.4,
    rotation_range=40,
    width_shift_range=0.4,
    height_shift_range=0.4,
    zoom_range=0.3,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=30.0,
    vertical_flip=True,
    fill_mode="nearest",
)
```

Cambiando algunos parámetros del DataAugmentation, haciendo que incremente de forma directa la cantidad de imágenes y variabilidad de las mismas, buscando una mayor generalización de los aprendizajes del modelo y evitar el sobreajuste.

Tras el cambio de estos parámetros, los resultados del entrenamiento han sido los siguientes:

```
Epoch 120: ReduceLROnPlateau reducing learning rate to 7.999999979801942e-07.
187/187 ━━━━━━━━━━━━━━━━━━━━ 47s 246ms/step - accuracy: 0.4101 - loss: 2.1076 - val_accuracy: 0.5405 - val_loss: 1.7086 - l
earning_rate: 4.0000e-06
```

En la época número 120, la última, los valores de aprendizaje han sido de un 41% y un valor de error del 2.10 en las imágenes de entrenamiento. Por otro lado, en las imágenes de validación, tenemos una precisión del 54% y un valor de error de 1.7, siendo estos valores algo mejores.

Este modelo volverá a ser entrenado buscando unos mejores valores del modelo, con la finalidad de tener la mejor precisión posible para este tipo de animales.

MODELO DE AVES SALVAJES

Al igual que en el modelo de los animales domésticos, en el modelo de aves, hemos vuelto a entrenar el modelo, cambiando algunos de los parámetros con respecto al primer entrenamiento de este modelo. Estos han sido los cambios realizados:

```
# Parámetros del modelo
epocas = 150  # Número de épocas
altura = 128  # Altura de las imágenes de entrada
longitud = 128  # Longitud de las imágenes de entrada
batch_size = 16  # Número de imágenes procesadas por lote
pasos = 4500 // batch_size  # Número de pasos por época
pasos_validacion = 1100 // batch_size  # Número de pasos para validación
clases = 8  # Número de clases (una para cada clase de animal)
lr = 0.0001  # Tasa de aprendizaje
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)
```

```
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.4,
    rotation_range=20,
    width_shift_range=0.4,
    height_shift_range=0.4,
    zoom_range=0.3,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=20.0,
    vertical_flip=True,
    fill_mode="nearest",
)
```

He cambiado el valor del batch_size, para que el modelo tenga menos imágenes por lote y por lo que el entrenamiento sea más rápido, y así, quizá los valores puedan mejorar, junto con la subida de valores del DataAugmentation, con el mismo propósito que el anterior, aumentar la cantidad de imágenes y generalizar el aprendizaje del modelo para evitar el sobreajuste de este.

Con el cambio de estos parámetros, hemos conseguido los siguientes resultados:

```
Epoch 111: ReduceLROnPlateau reducing learning rate to 3.199999980552093e-08.
281/281 ━━━━━━━━━━━━━━━━━━━━ 271s 964ms/step - accuracy: 0.4606 - loss: 1.5659 - val_accuracy: 0.5239 - val_loss: 1.3893 - le
arning_rate: 1.6000e-07
```

En la época número 111, el entrenamiento se ha parado y no ha conseguido acabar el entrenamiento, por lo que hemos perdido una cantidad de 39 épocas de entrenamiento. Aún así, el resultado ha sido de una precisión del 46% y un valor de pérdida del 1.56 en las imágenes de entrenamiento y del 52% de precisión, con el 1.38 de valor de pérdida en las imágenes de validación.

Estos datos se ajustan bastante más a lo que buscamos, pero por el hecho de haber perdido tantas épocas en el entrenamiento y por lo siguiente, podríamos haber perdido la mejora de los valores del modelo.

MODELO DE ANIMALES SALVAJES

El modelo de animales salvajes va a tener la misma configuración que los modelos anteriores, adaptando ciertos datos a los necesarios para este modelo. Estos son los cambios que tiene respecto a los demás:

```
# Parámetros del modelo
epocas = 150  # Número de épocas
altura = 128  # Altura de las imágenes de entrada
longitud = 128  # Longitud de las imágenes de entrada
batch_size = 32  # Número de imágenes procesadas por lote
pasos = 6000 // batch_size  # Número de pasos por época
pasos_validacion = 1800 // batch_size  # Número de pasos para validación
clases = 12  # Número de clases (una para cada clase de animal)
lr = 0.0001  # Tasa de aprendizaje
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)
```

```
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.3,
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.2,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=20.0,
    vertical_flip=True,
    fill_mode="nearest",
)
```

He adaptado el número de épocas a lo que veía mas conveniente, al igual que el numero de pasos, numero de pasos por validación, clases y batch_size.

Además en el DataAugmentation, he puesto los valores de la imagen para tener una variabilidad de imágenes contundentes, pero sin distorsionar la imagen y así, no confundir al modelo a la hora del entrenamiento.

El resultado del entrenamiento del modelo ha sido el siguiente:

```
Epoch 150/150
187/187 ━━━━━━━━━━━━━━━━━━━━ 51s 268ms/step - accuracy: 0.3965 - loss: 2.0393 - val_accuracy: 0.5064 - val_loss: 1.
7643 - learning_rate: 1.6000e-07
```

Tenemos en el entrenamiento unos datos de 39% de precisión y un 2.03 de valor de error, mientras que en la validación obtenemos un 50% de precisión y un 1.76 de valor de error. Este modelo volverá a ser entrenado debido a los pobres datos que hemos obtenido. Al igual que en los demás modelos, cambiaremos datos para ver si mejora.

MODELO DE REPTILES Y ANFIBIOS

Respecto al modelo de los reptiles, los valores de su entrenamiento fueron bastante adecuados y no estaba planteado más entrenamientos para este modelo. El problema surge cuando se comprueba su eficacia manualmente mediante este código:

```python
# Importamos librerías
import tensorflow as tf
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# Cargamos el modelo
modelo = tf.keras.models.load_model('../Reptiles/modeloReptiles.keras')

# Clases
clases = ['culebra', 'lagarto', 'rana', 'tortuga']

# Procesamos la imagen
def procesar_imagen(ruta, tamaño=(128, 128)):
    imagen = Image.open(ruta).convert('RGB')
    imagen = imagen.resize(tamaño)
    array = np.array(imagen) / 255.0
    array = np.expand_dims(array, axis=0)
    return array, imagen

# Predecimos la imagen
def predecir(ruta):
    array, imagen = procesar_imagen(ruta)
    pred = modelo.predict(array)
    clase = clases[np.argmax(pred)]
    probabilidad = np.max(pred)

    print(f'Predicción: {clase} ({probabilidad*100:.2f}%)')
    plt.imshow(imagen)
    plt.axis('off')
    plt.title(f'{clase} ({probabilidad*100:.2f}%)')
    plt.show()

# Llamamos a la función con la imagen
predecir('../Reptiles/rana.jpg')
```

Mediante este código, tratamos de poder coger una imagen que se corresponda a uno de los 4 animales para los que el modelo ha sido entrenado. Con esta imagen, lo que se va a hacer es predecir mediante el uso del archivo “modeloReptiles.keras” (archivo donde se encuentra el modelo entrenado), el animal que se reconoce en la imagen subida.

Al comprobar su funcionamiento mediante varias imágenes de distintos animales, el error en la predicción de las imágenes asciende de manera considerable, por lo que decidimos volver a entrenar el modelo cambiando los parámetros para intentar obtener una mejor precisión. Los cambios en el entrenamiento del modelo han sido los siguientes:

```python
# Transformación de las imágenes para entrenamiento (Data Augmentation)
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.4,
    rotation_range=40,
    width_shift_range=0.4,
    height_shift_range=0.4,
    zoom_range=0.3,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=30.0,
    vertical_flip=True,
    fill_mode="nearest",
)
```

Aumento de los parámetros en el DataAugmentation para un mayor número imágenes y variabilidad de las mismas para un mejor aprendizaje del modelo.

```python
cnn = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(5, activation='softmax')
])
```

Reducción del número de capas de neuronas durante el entrenamiento, ya que, al ser un set de imágenes pequeños (4 clases), es innecesario usar tantas capas de neuronas, ya que puede provocar un sobreajuste del modelo y genera muchos parámetros, produciendo acciones negativas en el entrenamiento del mismo. Así, se generará menos información en cada época y el entrenamiento será más rápido.

```python
epocas = 120  # Número de épocas
altura = 128  # Altura de las imágenes de entrada
longitud = 128  # Longitud de las imágenes de entrada
batch_size = 16  # Número de imágenes procesadas por lote
pasos = 6000 // batch_size  # Número de pasos por época
pasos_validacion = 1800 // batch_size  # Número de pasos para validación
clases = 5  # Número de clases (una para cada clase de animal)
lr = 0.00001  # Tasa de aprendizaje
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)
```

Reducción del valor de “learning_rate”, haciendo que el modelo se fije más en los detalles de cada imagen y el aprendizaje de este sea mayor.

ERROR ENCONTRADO

Antes de volver a entrenar el modelo de animales salvajes nuevamente, ha surgido un error a la hora de entrenar el modelo de reptiles anterior.

Este error, obliga a cambiar el código de todos los modelos (domésticos, aves, reptiles y salvajes) para que estos no provoquen problemas en un futuro cuando se implanten en los drones. El problema viene porque el número de clases que había configurado en el código era de 1 clase más que las realmente existentes en el directorio. Es decir, si el número de animales a diferenciar en el caso de los reptiles era de 4 distintos, el número de clases en el código era de 5, como se ha podido ver en las capturas del código anteriormente insertadas.

Este error se produce por la existencia de carpetas ocultas dentro del directorio donde se encuentra el set de imágenes con el que entrenamos al modelo.

Esto se puso en su momento, ya que el sistema reconocía siempre 1 carpeta más (la oculta) que la cantidad de animales a reconocer por cada modelo. Entonces, a la hora de entrenar el modelo, este reconocía que había una carpeta más de la que en realidad había, pensando que no iba a haber problema en un futuro cuando se fueran a reconocer animales.

Cuando se pone a prueba el modelo con el código de predicción anterior, el sistema produce un error, y es que el número de casos posibles que ponemos nosotros será igual al número de animales distintos (clases reales) que nosotros hayamos configurado, mientras que el número de casos posibles para el sistema siempre será de 1 más (clases reales + clase oculta, que el número de clases que nosotros hayamos puesto.

La solución a este problema es filtrar las carpetas que el sistema va a reconocer como válidas, evitando que el sistema coja todas las carpetas existentes en dicho directorio. Esto se ha realizado de la siguiente manera:

```
# Limpiar la sesión de Keras
K.clear_session()

# Directorios de imágenes
datos_entrenamiento = '../Reptiles/Entrenamiento'
datos_validacion = '../Reptiles/Validacion'

# Clases específicas que se desean usar
clases_validas = ['Culebra', 'Lagarto', 'Rana', 'Tortuga']

# Filtrar las clases en los directorios de entrenamiento y validación
def filtrar_clases(directorio, clases_validas):
    clases_encontradas = [clase for clase in os.listdir(directorio) if os.path.isdir(os.path.join(directorio, clase))]
    clases_filtradas = [clase for clase in clases_encontradas if clase in clases_validas]
    return clases_filtradas

# Obtener clases válidas en entrenamiento y validación
clases_entrenamiento = filtrar_clases(datos_entrenamiento, clases_validas)
clases_validacion = filtrar_clases(datos_validacion, clases_validas)

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="./logs", histogram_freq=1)
```

A la hora de configurar la ruta por la que el modelo obtendrá las imágenes con las que entrenará, se configurará también el número de clases válidas, siendo las que no estén configuradas ahí erróneas y haciendo que el sistema no las incluya dentro del entrenamiento.

Primero, hacemos una lista de los distintos tipos de clases (por nombre) que habrá en el entrenamiento.

Después, creamos la función que va a filtrar las clases en el entrenamiento, mediante la creación de una variable en la que almacenaremos todas las clases que el sistema encuentre dentro del directorio de las imágenes.

Por último, a partir de las clases encontradas dentro del directorio, cogeremos aquellas clases que se llamen igual que las que estén dentro del directorio, excluyendo las que tengan un nombre distinto al de alguna de la lista configurada al principio de la imagen.

Devolvemos la lista de clases filtradas.

Ahora, mediante el uso de variables, almacenamos las carpetas de entrenamiento y validación filtrada, evitando el uso de un mayor número de clases que el que realmente nos interesa.

```
# Carga de las imágenes de entrenamiento
imagen_entrenamiento = entrenamiento_datagen.flow_from_directory(
    datos_entrenamiento,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical',
    classes=clases_entrenamiento,
)

# Carga de las imágenes de validación
imagen_validacion = validacion_datagen.flow_from_directory(
    datos_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical',
    classes=clases_validacion,
)
```

Además, a la hora de cargar las imágenes de entrenamiento y validación, haciendo que se escojan los datos de las rutas configuradas en “datos_entrenamiento” y “datos_validacion”, pero que únicamente escoja las clases configuradas en “clases_entrenamiento” y “clases_validacion”, en dicho orden.

Haciendo esto en todos los modelos (domésticos, salvajes, reptiles y aves), evitaremos este error.

Ahora, pondremos a entrenar todos los modelos de nuevo, ya que, es posible, que excluyendo una carpeta basura oculta del set de imágenes, los valores y la eficacia del modelo mejoren.

DESARROLLO DE MODELO DE ANIMALES DOMÉSTICOS DEFINITIVO

Empecemos viendo el resultado del modelo de animales domésticos o de ganado. Hemos obtenido los siguientes resultados:

```
Epoch 120/120
187/187 ━━━━━━━━━━━━━━━━━━━━ 101s 532ms/step - accuracy: 0.3604 - loss: 2.1585 - val_accuracy: 0.5293 - val_loss:
1.7097 - learning_rate: 4.0000e-06
```

En la época 120, hemos tenido un porcentaje bastante escaso de precisión y de valor de error en las imágenes de entrenamiento, 36% y 2.15 respectivamente, y mejorando un poco en las imágenes de validación, estando estos en un casi 53% de precisión y un 1.7 de valor de error.

Durante el entrenamiento, podemos observar analizando los valores por épocas, que el modelo mejora notablemente sus valores al principio del entrenamiento, quedando este, estancado en las últimas casi 20 épocas.

Por lo que, los resultados de este entrenamiento son peores que el anterior, siendo inútiles los cambios realizados anteriormente. A partir de aquí, analicemos los cambios realizados y a realizar para intentar una mejora en el entrenamiento.

En la captura, podemos ver, que, durante el entrenamiento, el valor de aprendizaje no ha cambiado, mientras que el entrenamiento sí que se ha ido estancando, por lo que, el “early_stop” debería de haber actuado y haber reducido este valor. Arreglaremos esto para el próximo entrenamiento.

Los valores tocados para el nuevo entrenamiento ha sido un incremento del número de épocas de 120 a 150, una reducción de los valores del DataAugmentation, para evitar que la imagen se distorsione y confunda al modelo, además, de una reducción de la tasa de aprendizaje de 0.0001 a 0.00001.

```python
# Parámetros del modelo
epocas = 150  # Número de épocas
altura = 128  # Altura de las imágenes de entrada
longitud = 128  # Longitud de las imágenes de entrada
batch_size = 32  # Número de imágenes procesadas por lote
pasos = 6000 // batch_size  # Número de pasos por época
pasos_validacion = 1800 // batch_size  # Número de pasos para validación
clases = 14  # Número de clases (una para cada clase de animal)
lr = 0.00001  # Tasa de aprendizaje
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)
```

```python
# Transformación de las imágenes para entrenamiento (Data Augmentation)
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.2,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.15,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=20.0,
    vertical_flip=True,
    fill_mode="nearest",
)
```

```
Epoch 150/150
187/187 ━━━━━━━━━━━━━━━━━━━━ 98s 517ms/step - accuracy: 0.4521 - loss: 4.7247 - val_accuracy: 0.5905 - val_loss:
4.3656 - learning_rate: 1.0000e-05
```

En la época 150, el modelo ha tenido estos resultados:

Un 45% de precisión y un valor de error del 4.7 en el entrenamiento, y un 59% de precisión y un valor de error del 4.36 en la validación, siendo los valores de error un poco altos, mejorando de manera contraria los valores de precisión.

Entrenaremos de nuevo el modelo para intentar bajar el valor de error y subir el de precisión.

Hemos cambiado los siguientes datos:

```
# Transformación de las imágenes para entrenamiento (Data Augmentation)
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.3,
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.2,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=30.0,
    vertical_flip=True,
    fill_mode="nearest",
)
```

```
batch_size = 64  # Número de imágenes procesadas por lote
```

Hemos cambiado los valores de modificación del DataAugmentation, haciendo que haya algún tipo más de dato que anteriormente, junto con el batch_size, que hemos aumentado, para ver si esto le puede venir bien al modelo o sigue obteniendo el mismo valor de error que antes.

```
Epoch 150/150
93/93 ━━━━━━━━━━━━━━━━━━━━ 47s 488ms/step - accuracy: 0.3107 - loss: 6.5878 - val_accuracy: 0.5438 - val_loss: 6.
0608 - learning_rate: 1.0000e-05
```

El resultado del entrenamiento no ha sido satisfactorio, teniendo un resultado de 31% en precisión y un 6.58 de valor de error en el entrenamiento, con un 54% de precisión y un 6.06 de valor de error en la validación.

Seguiremos entrenando el modelo para intentar tener un mejor valor.

Para este siguiente entrenamiento lo único que hemos cambiado ha sido el valor del batch_size, que estaba en 64, siendo cambiado a 16, ya que como podemos comprobar, un valor tan grande no le viene bien al modelo:

```
batch_size = 16  # Número de imágenes procesadas por lote
```

Los resultados del entrenamiento han sido los siguientes:

```
Epoch 150/150
387/387 ━━━━━━━━━━━━━━━━━━━━ 41s 104ms/step - accuracy: 0.4997 - loss: 3.2304 - val_accuracy: 0.5817 - val_loss:
2.9909 - learning_rate: 1.0000e-05
```

Hemos podido obtener unos mejores números que lo que teníamos anteriormente, teniendo un casi 50% de precisión y un valor de error de 3.23 en el entrenamiento, con un 58% de precisión y un valor de error de 2.99 en la validación. Este entrenamiento se ajusta mucho más

a lo que buscamos, por lo que seguiremos entrenando para ver si podemos mejorar aún más estos valores.

Tras cambiar estos datos de entrenamiento:

```
epocas = 170  # Número de épocas
```
Ya que hemos observado que el entrenamiento en las últimas 10 épocas seguía mejorando, aunque de manera lenta.

```
lr = 0.0001  # Tasa de aprendizaje
```
Bajada de la tasa de aprendizaje de 0.00001 a 0.0001, para ver si podemos obtener un rango de mejora en este cambio.

```
channel_shift_range=30.0,
```
Subida del valor de desplazamiento de colores RGB en la imagen, para ver si afecta positiva o negativamente al aprendizaje.

```
Epoch 105: ReduceLROnPlateau reducing learning rate to 7.999999979801942e-07.
387/387 ━━━━━━━━━━━━━━━━━━━━ 208s 537ms/step - accuracy: 0.4489 - loss: 1.9623 - val_accuracy: 0.5676 - val_loss: 1.6192 - learning_
rate: 4.0000e-06
```

Los resultados del entrenamiento son mejores respecto a los anteriores:

En el entrenamiento tenemos un 44% de precisión y un valor de error de 1.96, mientras que en la validación obtenemos una precisión del 56% y un valor de error de 1.61.

Al ser de los mejores datos obtenidos en los últimos entrenamientos, no entrenaremos más este modelo. Los resultados definitivos del modelo de reconocimiento de animales domésticos son los siguientes:

Una precisión del 56% y un valor de error de 1.61.

DESARROLLO DE MODELO DE REPTILES Y ANFIBIOS

Respecto a los reptiles, el nuevo entrenamiento ha dejado unos buenos valores tanto en la precisión como en el valor de error. Han sido los siguientes:

```
Epoch 103: ReduceLROnPlateau reducing learning rate to 1.600000018697756e-07.
187/187 ━━━━━━━━━━━━━━━━━━━━ 199s 1s/step - accuracy: 0.6831 - loss: 0.9652 - val_accuracy: 0.7161 - val_loss: 0.87
64 - learning_rate: 8.0000e-07
```

El modelo ha parado en la época 103, por lo que no ha acabado el entrenamiento por causas que desconozco.

En cuanto a los valores que ha dado el modelo, tenemos en el entrenamiento una precisión del 68% y un valor de error del 0.96, siendo mejorado en la validación, con un 71% de precisión y un valor de error el 0.87.

Teniendo en cuenta el valor de error, un resultado de menos de 1 tanto en el entrenamiento como en la validación es realmente bueno, por lo que, procederemos a predecir imágenes con el modelo, para ver el resultado real del modelo.

La predicción del modelo es mejor que lo anterior, pero sigue fallando en algunas imágenes, por lo que pondremos a entrenar el modelo de nuevo, cambiando algún parámetro para poder aumentar la mejora del modelo.

Los cambios realizados son los siguientes:

```
# Transformación de las imágenes para entrenamiento (Data Augmentation)
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.3,
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.2,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=20.0,
    vertical_flip=True,
    fill_mode="nearest",
)
```

```
# Parámetros del modelo
epocas = 120  # Número de épocas
altura = 128  # Altura de las imágenes de entrada
longitud = 128  # Longitud de las imágenes de entrada
batch_size = 32  # Número de imágenes procesadas por lote
pasos = 2500 // batch_size  # Número de pasos por época
pasos_validacion = 1000 // batch_size  # Número de pasos para validación
clases = 5  # Número de clases (una para cada clase de animal)
lr = 0.00001  # Tasa de aprendizaje
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)
```

Reducción de los valores de DataAugmentation, para evitar una distorsión importante de la imagen, además de un aumento del batch_size del 16 al 32, además de la reducción de los pasos durante el entrenamiento (debido a la poca cantidad de las imágenes) y durante la validación, por la misma razón.

```
Epoch 120/120
78/78 ━━━━━━━━━━━━━━━━━━━━ 40s 507ms/step - accuracy: 0.5249 - loss: 2.0009 - val_accuracy: 0.5831 - val_loss: 1.93
56 - learning_rate: 1.0000e-05
```

El resultado del entrenamiento ha sido peor que los anteriores, siendo del 52% y un valor de 2.00 en el error, mientras que en la validación hemos obtenido un 58% de precisión y un 1.93 de error.

Son datos relativamente bajos, por lo que volveremos a entrenar el modelo de nuevo, cambiando los siguientes datos:

```
# Transformación de las imágenes para entrenamiento (Data Augmentation)
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.3,
    rotation_range=40,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.3,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=30.0,
    vertical_flip=True,
    fill_mode="nearest",
)
batch_size = 16  # Número de imágenes procesadas por lote
```

Hemos cambiado como solemos hacer en todos los entrenamientos, los valores del DataAugmentation, ya que es una parte fundamental en el entrenamiento que nos puede beneficiar tanto como perjudicar, por lo que hay que dar con los valores exactos, además de

reducir el batch_size, para obtener un mayor número de pasos durante las épocas con menos cantidad de información por época, cosa que puede venir bien al modelo.

Una vez cambiados los datos, entrenamos de nuevo el modelo. El resultado del entrenamiento ha sido el siguiente:

```
Epoch 120/120
156/156 ━━━━━━━━━━━━━━━━━━━━ 69s 442ms/step - accuracy: 0.5684 - loss: 1.6654 - val_accuracy: 0.5651 - val_loss: 1.
6393 - learning_rate: 1.0000e-05
```

Obtenemos un 56% de precisión y un valor de error de 1.66 en el entrenamiento, mientras que, en la validación, tenemos un 56% de precisión con un 1.63 de valor de error.

Observando el proceso de entrenamiento, podemos observar que este, hasta la época 120, sigue mejorando de manera progresiva, por lo que, si ampliamos el número de épocas, podemos llegar a conseguir unos mejores valores, y por consecuencia, una mejor precisión.

```
Epoch 113/120
107/156 ━━━━━━━━━━━━━━━━━━━━ 32s 668ms/step - accuracy: 0.5634 - loss: 1.7225
Epoch 113: saving model to modelo_checkpoint.keras
156/156 ━━━━━━━━━━━━━━━━━━━━ 141s 900ms/step - accuracy: 0.5643 - loss: 1.7160 - val_accuracy: 0.5609 - val_loss:
1.6700 - learning_rate: 1.0000e-05
Epoch 114/120
156/156 ━━━━━━━━━━━━━━━━━━━━ 84s 537ms/step - accuracy: 0.5748 - loss: 1.6711 - val_accuracy: 0.5554 - val_loss: 1.
6719 - learning_rate: 1.0000e-05
Epoch 115/120
156/156 ━━━━━━━━━━━━━━━━━━━━ 159s 1s/step - accuracy: 0.5573 - loss: 1.7106 - val_accuracy: 0.5679 - val_loss: 1.66
10 - learning_rate: 1.0000e-05
Epoch 116/120
156/156 ━━━━━━━━━━━━━━━━━━━━ 66s 423ms/step - accuracy: 0.5741 - loss: 1.6815 - val_accuracy: 0.5665 - val_loss: 1.
6566 - learning_rate: 1.0000e-05
Epoch 117/120
156/156 ━━━━━━━━━━━━━━━━━━━━ 169s 1s/step - accuracy: 0.5847 - loss: 1.6589 - val_accuracy: 0.5679 - val_loss: 1.64
83 - learning_rate: 1.0000e-05
Epoch 118/120
156/156 ━━━━━━━━━━━━━━━━━━━━ 79s 507ms/step - accuracy: 0.5686 - loss: 1.6784 - val_accuracy: 0.5706 - val_loss: 1.
6430 - learning_rate: 1.0000e-05
Epoch 119/120
156/156 ━━━━━━━━━━━━━━━━━━━━ 155s 993ms/step - accuracy: 0.5680 - loss: 1.6820 - val_accuracy: 0.5637 - val_loss:
1.6422 - learning_rate: 1.0000e-05
```

Ahí podemos ver la progresión positiva en cuanto al valor de error. A continuación, cambiaremos el entrenamiento a 170 épocas y procederemos a entrenar dicho modelo.

```
epocas = 170  # Número de épocas
```

El resultado del entrenamiento con el aumento de épocas ha sido el siguiente:

```
Epoch 170/170
156/156 ━━━━━━━━━━━━━━━━━━━━ 62s 393ms/step - accuracy: 0.5730 - loss: 1.4633 - val_accuracy: 0.6136 - val_loss: 1.
3676 - learning_rate: 1.0000e-05
```

Los resultados de entrenamiento han sido de 57% de precisión y 1.46 valor de error, mientras que en la validación tenemos un 61% de precisión con un 1.36 de valor de error.

Los cambios a realizar para el último entrenamiento del modelo serán los siguientes:

```
batch_size = 32  # Número de imágenes procesadas por lote
```
Subida de 16 a 32 el número de imágenes por lote.

```
Epoch 170/170
78/78 ━━━━━━━━━━━━━━━━━━━━ 70s 893ms/step - accuracy: 0.5396 - loss: 1.7052 - val_accuracy: 0.6233 - val_loss: 1.6423 - learning_rate:
1.0000e-05
```

El resultado de este último entrenamiento ha sido de un 53% de precisión y un valor de 1.70 de error en el entreno y una precisión de un 62% y un valor de error de 1.64 en la validación. Este ha sido el último entrenamiento del modelo de reptiles, ya que por más que variemos los parámetros de dicho modelo, la mejora de los valores es mínimo o nulo.

Por lo que, la precisión del modelo de reptiles es del 62% de precisión y un valor de error del 1.64.

DESARROLLO DE MODELO DE AVES SALVAJES

```
Epoch 111: ReduceLROnPlateau reducing learning rate to 3.1999999980552093e-08.
281/281 ━━━━━━━━━━━━━━━━━━━━ 271s 964ms/step - accuracy: 0.4606 - loss: 1.5659 - val_accuracy: 0.5239 - val_loss:
1.3893 - learning_rate: 1.6000e-07
```

En el modelo encargado del reconocimiento de las aves, podemos observar que este se ha parado en la época 111, por lo que no ha acabado el entrenamiento por causas que desconozco a día de hoy.

Los parámetros que ha dado el entrenamiento como resultado han sido pobres para el número de clases que este tiene que reconocer, siendo durante el entrenamiento de un 46% y un 1.56 en el valor de error, y en la validación, del 52% de precisión y de 1.38 de valor de error.

Cambiaremos algunos parámetros del modelo para ver si podemos mejorar de alguna forma los valores del modelo con la finalidad de conseguir una mayor eficacia. Estos son:

```
epocas = 150  # Número de épocas
altura = 128  # Altura de las imágenes de entrada
longitud = 128  # Longitud de las imágenes de entrada
batch_size = 32  # Número de imágenes procesadas por lote
pasos = 4500 // batch_size  # Número de pasos por época
pasos_validacion = 1100 // batch_size  # Número de pasos para validación
clases = 7  # Número de clases (una para cada clase de animal)
lr = 0.0001  # Tasa de aprendizaje
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)
```

```
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.3,
    rotation_range=20,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.2,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=20.0,
    vertical_flip=True,
    fill_mode="nearest",
)
```

Aumento del batch_size, para permitir un entrenamiento más estable y rápido, aprovechando mejor así la arquitectura de la GPU.

Además, reduciremos los valores del DataAugmentation, con la finalidad de no trastocar de manera importante la imagen y confundir al modelo.

Los resultados del entrenamiento del modelo serán los siguientes:

```
Epoch 136: ReduceLROnPlateau reducing learning rate to 6.399999818995639e-09.
140/140 ━━━━━━━━━━━━━━━━━━━━ 81s 564ms/step - accuracy: 0.4912 - loss: 1.5062 - val_accuracy: 0.5956 - val_loss: 1.
3030 - learning_rate: 3.2000e-08
```

Después de los cambios realizados en el entrenamiento, los resultados del mismo han mejorado de manera notable, obteniendo durante el entrenamiento un 49% de precisión y un valor de error de 1.50, mientras que en la validación hemos obtenido unos valores de casi un 60%, siendo un 1.30 de valor de error. El modelo volverá a ser entrenado para ver si se puede mejorar estas cifras.

```
batch_size = 16  # Número de imágenes procesadas por lote
```

Cambio de batch_size de 32 a 16

```
channel_shift_range=30.0,
```

Subida de valor de control de desplazamiento aleatorio de los valores de canales de color de la imagen para obtener una mayor variabilidad de imágenes.

Cambiando de nuevo algunos valores del entrenamiento, para ver si puede mejorar, vemos que el resultado del entrenamiento con los valores cambiados será el siguiente:

```
Epoch 150/150
70/70 ━━━━━━━━━━━━━━━━━━━━ 52s 718ms/step - accuracy: 0.4459 - loss: 1.5935 - val_accuracy: 0.5781 - val_loss: 1.34
99 - learning_rate: 8.0000e-07
```

Donde tenemos un 44% de precisión y un 1.59 de valor de error durante el entrenamiento, con un 57% de precisión y un 1.34 de valor de error durante la validación.

Vemos que los resultados de los entrenamientos por más que cambiemos valores y usemos alguna ayuda como puede ser modelos preentrenados, no conseguimos que mejore mucho más, llegando en algunos casos como el anterior a empeorar. Haremos un par de entrenamientos más para ver si podemos mejorar el modelo, pero puede que hayamos llegado al mejor resultado que se puede obtener para el modelo.

Los cambios realizados en el modelo han sido los siguientes:

`rotation_range=30,` Subimos el valor de rotación de la imagen en del DataAugmentation

`batch_size = 32  # Número de imágenes procesadas por lote` Subimos el número de imágenes procesadas por lote de 16 a 32, como estaba anteriormente, ya que un valor bajo no ha servido de mejora para el modelo.

```
Epoch 150/150
140/140 ━━━━━━━━━━━━━━━━━━━━ 64s 443ms/step - accuracy: 0.5309 - loss: 1.5349 - val_accuracy: 0.6011 - val_loss: 1.
2697 - learning_rate: 1.6000e-07
```

Los resultados del entrenamiento han sido notablemente mejores que los anteriores, pero vemos que tampoco mejoran de manera exagerada aun cambiando bastantes variables que deberían de favorecer el entrenamiento del mismo.

Estos resultados han sido un 53% de precisión con un valor de error de 1.53 en el entrenamiento, mientras que, en la validación del modelo, hemos obtenido un 60% de precisión y un 1.26 de valor de error.

Al ver que los modelos no mejoran de manera considerable, hemos decidido dejar estos valores como los definitivos para la detección de animales salvajes, siendo de 60% de precisión y un 1.26 de valor de error.

DESARROLLO DE MODELO DE ANIMALES SALVAJES

Volvemos a entrenar el modelo de animales salvajes. Esta vez, no cambiamos ningún valor, únicamente el número de clases al que hacemos referencia, resolviendo el error anterior.

El resultado ha sido el siguiente:

```
Epoch 123: ReduceLROnPlateau reducing learning rate to 1.6000000018697756e-07.
187/187 ━━━━━━━━━━━━━━━━━━━━ 482s 3s/step - accuracy: 0.3971 - loss: 2.0504 - val_accuracy: 0.5022 - val_loss: 1.76
40 - learning_rate: 8.0000e-07
```

El entrenamiento se ha parado por causas que desconocemos en la época número 123, cuando debería de haber llegado al número 150.

Los resultados que obtenemos han sido parecidos a los anteriores, teniendo un 39% de precisión y un 2.05 de valor de error en el entrenamiento, mientras que tenemos un 50% de precisión y un 1.76 de valor de error en la validación.

El modelo cambiará datos de entrenamiento y se volverá a entrenar para mejorar sus datos.

```python
# Transformación de las imágenes para entrenamiento (Data Augmentation)
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalización de las imágenes
    shear_range=0.3,
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.2,  # Incrementar el rango de zoom
    horizontal_flip=True,
    brightness_range=[0.7, 1.3],
    channel_shift_range=20.0,
    vertical_flip=True,
    fill_mode="nearest",
)
```

Hemos cambiado los datos del DataAugmentation para ver si variando este tipo de datos podemos obtener una mejora en el modelo.

Los resultados del entrenamiento han sido los siguientes.

```
Epoch 130: ReduceLROnPlateau reducing learning rate to 1.6000000018697756e-07.
187/187 ━━━━━━━━━━━━━━━━━━━━ 47s 247ms/step - accuracy: 0.3916 - loss: 2.0092 - val_accuracy: 0.4952 - val_loss: 1.
7624 - learning_rate: 8.0000e-07
```

Tenemos un 39% de precisión y un 2.00 de valor de error en el entrenamiento, mientras que en la validación obtenemos un 49% de precisión y un valor de error de 1.76.

Volveremos a entrenar el modelo.

Cambiando los siguientes valores:

```python
channel_shift_range=30.0,
```

```python
batch_size = 16  # Número de imágenes procesadas por lote
```

Hemos ampliado el valor que desplaza el desplazamiento aleatorio de los valores de los canales de color para obtener una mayor variabilidad de imágenes y disminuido el batch_size (número de imágenes por lote), para ampliar la cantidad de pasos y pasos de validación por entrenamiento y que el modelo entrene de manera más detenida.

El resultado ha sido el siguiente:

```
Epoch 119: ReduceLROnPlateau reducing learning rate to 3.199999980552093e-08.
375/375 ━━━━━━━━━━━━━━━━━━━━ 215s 573ms/step - accuracy: 0.3859 - loss: 2.0476 - val_accuracy: 0.4767 - val_loss: 1.8257 - learning_rat
e: 1.6000e-07
```

El entrenamiento se ha parado en la época número 119, en vez de la época 150 por una causa

desconocida, teniendo unos resultados finales de una precisión del 38% y un valor de error de 2.04 en el entrenamiento y en la validación, una precisión del 47% y un valor de error de 1.87.

Cambiando este valor:

```
batch_size = 32  # Número de imágenes procesadas por lote
```

Hemos aumentado el batch_size de 16 a 32, ya que el anterior no ha favorecido a la mejora del modelo.

Entrenamos por última vez este modelo:

```
Epoch 121: ReduceLROnPlateau reducing learning rate to 1.6000000186977566e-07.
187/187 ━━━━━━━━━━━━━━━━━━━━ 199s 1s/step - accuracy: 0.4033 - loss: 2.0506 - val_accuracy: 0.5054 - val_loss: 1.7746 - learning_rate:
8.0000e-07
```

El último entrenamiento de el modelo de IA para el reconocimiento de animales salvajes ha tenido una precisión del 40% y un valor de error de 2.05 en e entrenamiento y un 50% de precisión y 1.77 de valor de error durante la validación.

El modelo no va a ser entrenado más ya que por más que cambiamos parámetros de dicho entrenamiento, al igual que en los demás modelos, estos no mejoran notablemente, por lo que los valores finales del modelo que reconocerá los animales salvajes de la comunidad de Castilla y León tendrá una precisión del 50% y un valor de error de 1.77.

CÓDIGO PARA RECONOCER LAS ZONAS DE CALOR DE UNA IMAGEN

Por último, se ha realizado un código encargado de recortar una imagen RGB (imagen a color), a partir del reconocimiento de una zona de calor en una imagen térmica mediante la detección de coordenadas de la zona de calor, siendo usadas para recortar la imagen RGB.

Este código va a ayudar a los modelos de IA anteriores para evitar que reconozcan zonas de la imagen innecesarias y centrarse en reconocer la parte fundamental, que es el animal, haciendo así, que no se reconozcan por error elementos como árboles, arbustos o rocas.

El código aportado es el siguiente:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Importamos las librerías y módulos que se van a utilizar en el código posteriormente.

- CV2: importada como librería para el procesamiento de imágenes (detección de objetos, aplicación de filtros y modificación de imágenes).
- Numpy: librería utilizada para el trabajo con álgebra, cuentas matemáticas…
- Matplotlib: librería para visualizar los datos en Python (imágenes) y eliminar los ejes de las imágenes.

```
def modelo_inteligente_zona_calor(ruta_imagen_thermal, ruta_imagen_rgb):
    # Leer imagen térmica y convertir a HSV
    imagen_termica = cv2.imread(ruta_imagen_termica)
    imagen_rgb = cv2.imread(ruta_imagen_rgb)

    imagen_termica_hsv = cv2.cvtColor(imagen_termica, cv2.COLOR_BGR2HSV)

    # Rango para rojo
    rojo_bajo1 = np.array([0, 100, 100])
    rojo_alto1 = np.array([10, 255, 255])
    rojo_bajo2 = np.array([160, 100, 100])
    rojo_alto2 = np.array([179, 255, 255])

    # Rango para naranja
    naranja_bajo = np.array([11, 100, 100])
    naranja_alto = np.array([25, 255, 255])

    # Crear máscaras para detectar las zonas de calor (rojo y naranja)
    mascara_rojo = cv2.bitwise_or(
        cv2.inRange(imagen_termica_hsv, rojo_bajo1, rojo_alto1),
        cv2.inRange(imagen_termica_hsv, rojo_bajo2, rojo_alto2)
    )
    mascara_naranja = cv2.inRange(imagen_termica_hsv, naranja_bajo, naranja_alto)
    mascara_total = cv2.bitwise_or(mascara_rojo, mascara_naranja)

    # Detección de contornos en la imagen térmica
    contornos, _ = cv2.findContours(mascara_total, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Hemos hecho una función que se encarga de detectar la zona de calor y posteriormente, recortar la imagen RGB a partir de las coordenadas reconocidas en la zona de calor.

Primero, cargamos las imágenes desde el disco (la térmica y la RGB) y la lee en formato BGR (azul, verde y rojo).

Después, convertimos la imagen térmica a formato HS, que facilita la detección de colores específicos como el rojo o el naranja.

Más tarde, definimos la saturación de los colores rojo y naranja, para saber por qué rangos tiene que moverse la detección de los colores. Configuraremos el color mínimo a reconocer y el máximo del naranja y el rojo. Utilizamos dos rangos altos y bajos en el rojo para reconocer toda la gama de rojos que pueden existir en la imagen (pasamos el espectro del rojo), mientras que del naranja únicamente reconocemos los tonos más oscurecidos.

A continuación, creamos las máscaras para detectar las zonas de calor (rojo y naranja). Estas máscaras, harán que los pixeles en los que tengamos los colores configurados se pongan en blanco, y los píxeles que no contengan estos colores, se pondrán en negro. Con la función “bitwise”, unimos ambas máscaras para crear una única máscara que englobe ambos colores.

Por último, detectamos los contornos más externos de la máscara creada anteriormente.

```
if contornos:
    x, y, w, h = cv2.boundingRect(np.vstack(contornos))

    # Recortar la imagen térmica para mostrar la zona de calor
    imagen_termica_recortada = imagen_termica[y:y+h, x:x+w]

    # Recortar la imagen RGB utilizando las mismas coordenadas de la imagen térmica
    imagen_rgb_recortada = imagen_rgb[y:y+h, x:x+w]

    # Mostrar los resultados
    plt.subplot(1, 2, 1)
    plt.title("Imagen Térmica Recortada")
    plt.imshow(cv2.cvtColor(imagen_termica_recortada, cv2.COLOR_BGR2RGB))
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.title("Imagen RGB Recortada")
    plt.imshow(cv2.cvtColor(imagen_rgb_recortada, cv2.COLOR_BGR2RGB))
    plt.axis("off")

    plt.show()

    return (x, y, w, h), imagen_rgb_recortada
else:
    print("No se detectó zona de calor.")
    return None, None
```

Si hay un contorno que existe, es decir, hay zonas de calor en la imagen, se crea un rectángulo que delimitará dichas zonas. Posteriormente, se recortará la imagen térmica y RGB acorde con el rectángulo delimitador o coordenadas correspondientes.

Por último, mostramos los resultados, con ambas imágenes recortadas y quitando los ejes de las imágenes, ya que son inútiles en este caso.

La función subplot, se utiliza para organizar las imágenes a la hora de mostrarlas en la "cuadricula", habiendo una única fila y poniendo una imagen al lado de la otra.

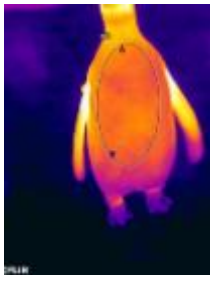
Si no se encuentra un contorno y la zona de calor no existiese, saltará un mensaje de error en el que diremos que no se ha mostrado ninguna zona de calor y devolveremos valores nulos a la función.

```
coordenadas, recorte_rgb = modelo_inteligente_zona_calor("imagentermica.png", "imagenrgb.png")
print("Coordenadas:", coordenadas)
```

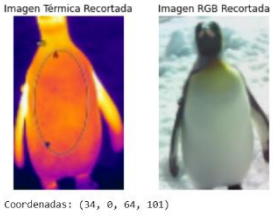
Por último, llamamos a la función realizada y mostramos las coordenadas del "rectángulo delimitador".

Los resultados del código son los siguientes.

Pasando estas imágenes al script:



Nos devuelve lo siguiente:



Las imágenes recortadas, centrándonos en el punto de calor de la imagen, por lo que podemos decir que el script funciona de manera correcta.