# Proyecto Animalia AI

Reconocimiento de Imágenes con Inteligencia Artificial


**Integrantes:**

Loya Cadena Bryan Eduardo
Simbaña Rodriguez Kevin Bladimir
Tapia Rea Freddy Xavier
Trujillo Vistin Dennis Adrian


**Fecha:** 4 de febrero de 2025


*Universidad Central del Ecuador*
*Facultad de Ingeniería y Ciencias Aplicadas*
*Cátedra de Inteligencia Artificial*

**Resumen**

El proyecto *Animalia IA*, desarrollado por el Grupo 7, tiene como objetivo el diseño e implementación de un sistema avanzado de reconocimiento de imágenes basado en inteligencia artificial, específicamente utilizando redes neuronales convolucionales (CNNs). Este sistema está orientado a la clasificación de imágenes de diversas especies animales, seleccionadas por su relevancia en educación, divulgación científica y conservación de fauna. Las categorías incluyen, entre otras, perro, gato, vaca, lagarto, panda y chinchilla. El enfoque del proyecto abarca desde el preprocesamiento de datos y el entrenamiento supervisado, hasta la validación y evaluación del modelo mediante métricas estándar como precisión, recall y F1-score.

Para la creación y entrenamiento del modelo, se emplearon herramientas como *TensorFlow/Keras*, mientras que la visualización de métricas fue facilitada mediante *TensorBoard*. La validación del rendimiento del modelo se realizó a través de la matriz de confusión, lo que permitió obtener un alto nivel de precisión en la clasificación. Además, el proyecto integra un sistema de despliegue que utiliza las tecnologías *Next.js* y *NestJS*, desarrollando una interfaz gráfica interactiva que permite a los usuarios cargar imágenes y recibir predicciones de manera accesible y en tiempo real.

El sistema no solo está diseñado para su uso práctico, sino también para su evolución futura, permitiendo la integración de nuevos datos y la actualización continua de los modelos. Los resultados obtenidos validan la viabilidad del sistema en escenarios controlados, y este proyecto establece una base sólida para futuras investigaciones y aplicaciones del reconocimiento de imágenes en diversos campos.

# Índice


**1 Introducción** **4**
*1.1 Contexto del Proyecto* . . . 4
1.1.1 Importancia y Aplicaciones del Sistema Desarrollado . . . 4
*1.2 Objetivo General* . . . 4
*1.3 Objetivos Específicos* . . . 4
*1.4 Alcance del Proyecto* . . . 5

**2 Fundamentación Teórica** **5**
*2.1 Conceptos Clave* . . . 5
*2.2 Herramientas y Tecnologías* . . . 6
*2.3 Dataset* . . . 6
2.3.1 Características del Dataset . . . 7
2.3.2 Selección de Clases . . . 7
2.3.3 Preprocesamiento de los Datos . . . 7
2.3.4 Aplicaciones y Relevancia del Dataset . . . 8

**3 Diseño e Implementación** **8**
*3.1 Arquitectura General del Sistema* . . . 8
*3.2 Entrenamiento del Modelo* . . . 8
3.2.1 Listar todas las clases disponibles . . . 9
3.2.2 Configuraciones iniciales y preparación de datos . . . 9
3.2.3 Configuración del Modelo . . . 10
3.2.4 Entrenamiento del Modelo . . . 12
3.2.5 Evaluación del Modelo Entrenado . . . 14
3.2.6 Matriz de Confusión . . . 15

**4 Resultados del Modelo** **17**
*4.1 Desempeño del Modelo* . . . 18
*4.2 Visualización de Métricas* . . . 19
*4.3 Ejemplos de Predicción* . . . 21
*4.4 Análisis de Errores* . . . 22
*4.5 Conclusión de Resultados* . . . 23

**5 Interfaz Gráfica UI** **23**
*5.1 Elección de Next.js* . . . 24
*5.2 Elección de NestJS* . . . 24
*5.3 Elección de shadcn/ui* . . . 24
*5.4 Beneficios de la Combinación Tecnológica* . . . 25
*5.5 Diagrama Estructura de Directorios* . . . 25
*5.6 Estructura del Proyecto* . . . 26
5.6.1 Archivos de Configuración del Proyecto . . . 26
5.6.2 Carpeta `public/` . . . 26
5.6.3 Carpeta `app/` . . . 26
5.6.4 Carpeta `components/` . . . 27
5.6.5 Carpeta `hooks/` . . . 27
5.6.6 Carpeta `lib/` . . . 27
5.6.7 Carpeta `server/` . . . 27

5.6.8 Archivos de Paquetes . . . . . 27
5.6.9 Archivos de Caché y Build . . . . . 28
*5.7 Conversión del Modelo y Gestión de Archivos* . . . . . 28
5.7.1 Proceso de Conversión del Modelo . . . . . 28
*5.8 Conversión del Modelo a Formato TensorFlow.js* . . . . . 28
5.8.1 Código de Conversión . . . . . 28
5.8.2 Descripción del Código . . . . . 29
5.8.3 Salida en Consola . . . . . 29
5.8.4 Archivos Generados . . . . . 30
5.8.5 Ventajas del Formato JSON . . . . . 30
5.8.6 Integración en la Interfaz Gráfica . . . . . 30
5.8.7 Estructura Final de los Archivos del Modelo . . . . . 31

**6 Arquitectura y Diagramas de Despliegue 31**
*6.1 Arquitectura General* . . . . . 31
6.1.1 Componentes de la Arquitectura . . . . . 31
*6.2 Diagrama de Arquitectura del Sistema* . . . . . 32
*6.3 Diagrama de Despliegue* . . . . . 32
*6.4 Flujo de Trabajo del Sistema* . . . . . 33
*6.5 Beneficios del Enfoque de Despliegue* . . . . . 34
*6.6 Despliegue en Vercel* . . . . . 34
6.6.1 Ventajas de Utilizar Vercel . . . . . 34
6.6.2 Proceso de Despliegue en Vercel . . . . . 35
6.6.3 Consideraciones de Configuración . . . . . 35
6.6.4 Monitoreo y Mantenimiento . . . . . 36

**7 Conclusiones y Recomendaciones 36**
*7.1 Conclusiones* . . . . . 36
*7.2 Limitaciones* . . . . . 37
*7.3 Recomendaciones* . . . . . 37

**8 Referencias Bibliograficas 38**
*8.1 Modelos de Redes Neuronales Convolucionales (CNN) y Entrenamiento de Modelos de IA* . . . . . 38
*8.2 TensorFlow y TensorFlow.js* . . . . . 39
*8.3 Desarrollo de Interfaces de Usuario (UI)* . . . . . 39
*8.4 Despliegue en Vercel* . . . . . 39
*8.5 General* . . . . . 40

**A Anexos 40**

# 1. Introducción

## 1.1. Contexto del Proyecto

El reconocimiento de imágenes constituye una de las áreas más relevantes y avanzadas dentro del campo de la inteligencia artificial, enfocándose en la interpretación y análisis de datos visuales mediante técnicas computacionales. Este ámbito ha experimentado un crecimiento exponencial en las últimas décadas, principalmente debido al desarrollo de algoritmos de aprendizaje profundo y la disponibilidad de conjuntos de datos de gran escala. Su impacto es ampliamente visible en sectores como la salud, donde se utiliza para el diagnóstico médico; la seguridad, en sistemas de vigilancia y control de acceso; y el entretenimiento, en aplicaciones de realidad aumentada y virtual. En este contexto, el presente proyecto se posiciona como una iniciativa dirigida a la clasificación automatizada de imágenes de especies animales mediante redes neuronales convolucionales (CNNs), aprovechando su capacidad para identificar patrones visuales complejos.

### 1.1.1. Importancia y Aplicaciones del Sistema Desarrollado

El sistema de reconocimiento de imágenes desarrollado en este proyecto tiene un amplio potencial de aplicación, especialmente en los siguientes ámbitos:

- **Identificación automática de especies animales:** La capacidad de clasificar imágenes de distintas especies animales permite optimizar procesos en campos como la biología, la conservación de fauna y la gestión de biodiversidad. Este sistema minimiza la intervención humana y acelera el análisis de datos visuales, proporcionando resultados precisos en menor tiempo.

- **Educación y divulgación científica:** Mediante la integración de una interfaz interactiva, el sistema se convierte en una herramienta didáctica para instituciones educativas, museos y programas de sensibilización ambiental. Los usuarios pueden explorar y aprender sobre características específicas de diversas especies animales de forma práctica e intuitiva.

## 1.2. Objetivo General

El objetivo principal de este proyecto es diseñar e implementar un sistema avanzado de reconocimiento de imágenes basado en redes neuronales convolucionales, que garantice un alto nivel de precisión en la clasificación de imágenes pertenecientes a seis clases específicas de animales, seleccionadas por su relevancia en educación y divulgación científica. Este sistema busca ser eficiente, accesible y escalable, facilitando su uso en diversas aplicaciones prácticas.

## 1.3. Objetivos Específicos

Para alcanzar el objetivo general, se plantearon los siguientes objetivos específicos:

1. Desarrollar un modelo de red neuronal convolucional (CNN) entrenado con un subconjunto cuidadosamente seleccionado del dataset *Image Classification*

- *64 Classes - Animal*, asegurando una adecuada representación de las clases animales objetivo.

2. Validar el modelo utilizando métricas estándar de evaluación como precisión, recall y F1-score, garantizando un nivel de confianza igual o superior al 99 % en las predicciones realizadas por el sistema.

3. Diseñar e implementar una interfaz gráfica interactiva utilizando tecnologías modernas como *Next.js* y *NestJS*, permitiendo a los usuarios cargar imágenes y obtener predicciones en tiempo real de forma sencilla y eficiente.

4. Documentar exhaustivamente los resultados obtenidos, incluyendo el desempeño del modelo y las limitaciones identificadas, así como proponer estrategias de mejora y ampliación para futuras investigaciones.

### 1.4. Alcance del Proyecto

El alcance del proyecto abarca las siguientes actividades:

- El diseño, desarrollo y entrenamiento de un modelo de redes neuronales convolucionales, optimizado para la clasificación de seis clases animales específicas.

- La validación del modelo en un entorno controlado, empleando métricas estándar y herramientas de visualización como *TensorBoard*, para garantizar su robustez y confiabilidad.

- La implementación de un prototipo funcional accesible mediante una interfaz gráfica interactiva, capaz de realizar predicciones en tiempo real y facilitar la interacción con los usuarios finales.

Por otro lado, el proyecto excluye los siguientes aspectos:

- El diseño y la implementación de hardware especializado para la aceleración del procesamiento, como unidades de procesamiento gráfico (GPUs) dedicadas.

- La evaluación del modelo en escenarios no controlados o utilizando datos fuera del conjunto de entrenamiento y validación previamente definido.

## 2. Fundamentación Teórica

### 2.1. Conceptos Clave

El reconocimiento de imágenes, como subdisciplina de la inteligencia artificial, integra múltiples conceptos fundamentales que resultan indispensables para su desarrollo. A continuación, se describen los conceptos más relevantes para este proyecto:

- **Redes Neuronales Convolucionales (CNNs):** Las redes neuronales convolucionales son una arquitectura específica de redes neuronales profundas, diseñada para procesar datos con una estructura de cuadrícula, como las imágenes. Su principal característica radica en la capacidad de extraer características jerárquicas, desde patrones básicos como bordes y texturas hasta representaciones más complejas, gracias a la aplicación de operaciones de convolución y capas subsiguientes de agrupación (*pooling*).

- **Aprendizaje Supervisado:** El aprendizaje supervisado es un paradigma en el cual el modelo es entrenado utilizando un conjunto de datos etiquetados, es decir, donde cada entrada (imagen) tiene asociada una etiqueta correspondiente a su clase. Este enfoque permite al modelo aprender a generalizar patrones y realizar predicciones precisas sobre datos no vistos previamente.

- **Métricas de Evaluación:** Para evaluar el rendimiento del modelo, se emplean métricas estándar ampliamente utilizadas en el campo del aprendizaje automático:

  - **Precisión (*Precision*):** Proporción de predicciones correctas entre todas las predicciones positivas realizadas.
  - **Exhaustividad (*Recall*):** Capacidad del modelo para identificar correctamente todas las instancias positivas.
  - **F1-Score:** Media armónica entre precisión y exhaustividad, útil en escenarios donde es necesario equilibrar ambos factores.

## 2.2. Herramientas y Tecnologías

El desarrollo del sistema se apoyó en un conjunto de herramientas y tecnologías especializadas, seleccionadas por su eficacia y amplia adopción en la comunidad científica y tecnológica:

- **TensorFlow/Keras:** TensorFlow es un framework de código abierto diseñado para el desarrollo de modelos de aprendizaje profundo. Keras, su API de alto nivel, facilita la creación e implementación de redes neuronales mediante una interfaz intuitiva y modular.

- **TensorBoard:** Herramienta de visualización integrada en TensorFlow que permite monitorear en tiempo real el progreso del entrenamiento del modelo. Proporciona gráficos y análisis detallados de métricas como la pérdida, la precisión y la evolución de las capas internas del modelo.

- **Next.js y NestJS:** Next.js es un framework de React que permite construir interfaces de usuario dinámicas y de alto rendimiento, mientras que NestJS es un framework de backend basado en Node.js que facilita la creación de APIs robustas y escalables. Ambas tecnologías se utilizaron para desarrollar una interfaz gráfica interactiva y eficiente que permite la interacción con el modelo de inteligencia artificial.

## 2.3. Dataset

El conjunto de datos empleado en este proyecto proviene de la plataforma Kaggle y lleva por título *Image Classification - 64 Classes - Animal*. Este dataset, ampliamente reconocido en la comunidad de inteligencia artificial, contiene imágenes organizadas en 64 clases diferentes de animales. Su última actualización ocurrió hace cinco meses, y se encuentra estructurado de manera jerárquica para facilitar su manejo en tareas de clasificación multiclase y entrenamiento de modelos de redes neuronales profundas.

### 2.3.1. Características del Dataset

El dataset presenta las siguientes propiedades principales:

- **Total de Clases:** 64 clases de animales, cada una representada en una carpeta independiente. Las carpetas contienen imágenes que corresponden exclusivamente a una categoría específica.
- **Formato de las Imágenes:** Las imágenes están en formato PNG y tienen una resolución original de 512x512 píxeles.
- **Estructura:** La organización jerárquica del dataset permite una fácil identificación y segmentación de las clases individuales, lo cual es fundamental para su preprocesamiento y uso en sistemas de clasificación supervisada.

### 2.3.2. Selección de Clases

Con el objetivo de adaptar el conjunto de datos a los requerimientos específicos del proyecto, se realizó una selección estratégica de seis clases animales: **perro, gato, vaca, lagarto, chinchilla y panda**. Estas categorías fueron escogidas por su diversidad visual, su relevancia en el ámbito educativo y científico, y su capacidad para representar patrones visuales distintivos. La selección asegura una muestra balanceada que permite evaluar el desempeño del modelo en la clasificación de características visuales variadas.

### 2.3.3. Preprocesamiento de los Datos

El preprocesamiento del conjunto de datos fue un paso crucial para garantizar un rendimiento óptimo del modelo y facilitar su entrenamiento. Las etapas implementadas fueron las siguientes:

- **Filtrado de Clases:** Se extrajeron exclusivamente las imágenes correspondientes a las seis clases seleccionadas, reduciendo el tamaño del dataset original y enfocando el modelo en las categorías de interés.
- **Redimensionamiento:** Las imágenes fueron redimensionadas de su resolución original de 512x512 píxeles a un formato uniforme de 250x250 píxeles. Este ajuste redujo la complejidad computacional y estandarizó las entradas para la red neuronal.
- **Normalización:** Los valores de los píxeles de cada imagen fueron escalados al rango [0, 1], optimizando el proceso de aprendizaje del modelo al facilitar la convergencia durante el entrenamiento.
- **División del Dataset:** El conjunto de datos preprocesado se dividió en tres subconjuntos principales:
    - 80 % para entrenamiento, empleado en el aprendizaje supervisado del modelo.
    - 10 % para validación, utilizado para monitorear el desempeño del modelo y ajustar hiperparámetros, previniendo problemas como el sobreajuste (*overfitting*).

- 10 % para pruebas, reservado para evaluar la precisión, robustez y capacidad de generalización del modelo en datos no vistos previamente.

#### 2.3.4. Aplicaciones y Relevancia del Dataset

El dataset *Image Classification - 64 Classes - Animal* es una herramienta versátil con diversas aplicaciones en el ámbito de la inteligencia artificial, tales como:

- Clasificación multiclase de imágenes mediante sistemas de visión por computadora.
- Entrenamiento de redes neuronales convolucionales para identificar patrones visuales específicos asociados a diferentes especies animales.
- Desarrollo de aplicaciones educativas para la identificación automática de animales en entornos interactivos.
- Creación de tutoriales académicos y participación en competencias de clasificación de imágenes.

En el contexto de este proyecto, el conjunto de datos constituye la base para el desarrollo de un sistema de reconocimiento de imágenes de alta precisión. Las etapas de preprocesamiento, combinadas con la selección estratégica de clases, aseguran un entorno ideal para la evaluación rigurosa del modelo, promoviendo resultados confiables y representativos.

## 3. Diseño e Implementación

### 3.1. Arquitectura General del Sistema

El sistema de reconocimiento de imágenes se estructura en varios componentes principales:

- **Entrenamiento del Modelo:** Uso de CNNs para aprender patrones a partir de las imágenes.
- **Validación y Pruebas:** Evaluación del modelo con datos separados para medir su rendimiento.
- **Predicción:** Implementación de un módulo para realizar inferencias en nuevas imágenes.
- **Interfaz de Usuario:** Plataforma gráfica para facilitar la interacción con el sistema.

### 3.2. Entrenamiento del Modelo

El modelo fue entrenado utilizando TensorFlow/Keras con los siguientes pasos principales:

1. Carga y preprocesamiento del conjunto de datos seleccionado.
2. Definición de la arquitectura de la red neuronal.

3. Configuración del optimizador, función de pérdida y métricas de evaluación.
4. Entrenamiento con monitoreo mediante TensorBoard.

### 3.2.1. Listar todas las clases disponibles

**Explicación:** Este bloque de código se encarga de listar las clases disponibles en el conjunto de datos para que el usuario pueda ver qué categorías se van a usar en el modelo. El código usa la función `os.listdir` para leer las carpetas dentro del directorio base, que corresponden a las clases en el dataset.

**Código:**

```
import os

base_dir = '/kaggle/input/image-classification-64-classes-animal/image'
available_classes = os.listdir(base_dir)

print("Clases disponibles en el dataset:")
for i, class_name in enumerate(available_classes, 1):
    print(f"{i}. {class_name}")
```

**Explicación del código:**

- `base_dir`: Directorio base que contiene las imágenes.
- `os.listdir(base_dir)`: Lee todas las carpetas en el directorio base, cada una representando una clase.
- `enumerate`: Itera a través de las clases disponibles y las enumera.
- El resultado es una lista de clases que se imprimirá en consola.

**Salida esperada:**

```
Clases disponibles en el dataset:
1. chinchilla
2. lizard
3. cat
4. panda
5. cow
6. dog
```

### 3.2.2. Configuraciones iniciales y preparación de datos

**Explicación:** Este bloque prepara el conjunto de datos seleccionado, que incluye la separación en conjuntos de entrenamiento, validación y prueba. Se utilizan generadores de imágenes para aplicar aumentos de datos y normalizar las imágenes para que puedan ser alimentadas al modelo.

```
import os
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, classification_report
    , ConfusionMatrixDisplay
```

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split

base_dir = '/kaggle/input/image-classification-64-classes-animal/image'
selected_classes = ['chinchilla', 'lizard', 'cat', 'panda', 'cow', 'dog']
print(f"Clases seleccionadas: {selected_classes}")

# Cargar las im genes y sus etiquetas
image_paths = []
labels = []
for class_name in selected_classes:
    class_dir = os.path.join(base_dir, class_name)
    for image_name in os.listdir(class_dir):
        image_paths.append(os.path.join(class_dir, image_name))
        labels.append(class_name)

df = pd.DataFrame({'image_path': image_paths, 'label': labels})

# Dividir los datos en entrenamiento, validaci n y prueba
train_df, test_df = train_test_split(df, test_size=0.2, stratify=df['label'], random_state=42)
train_df, val_df = train_test_split(train_df, test_size=0.2, stratify=train_df['label'], random_state=42)

# Generadores de datos
train_datagen = ImageDataGenerator(rescale=1./255, rotation_range=20, width_shift_range=0.2, height_shift_range=0.2)
val_test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_dataframe(
    train_df, x_col='image_path', y_col='label', target_size=(224, 224), batch_size=32, class_mode='categorical'
)
val_generator = val_test_datagen.flow_from_dataframe(
    val_df, x_col='image_path', y_col='label', target_size=(224, 224), batch_size=32, class_mode='categorical'
)
test_generator = val_test_datagen.flow_from_dataframe(
    test_df, x_col='image_path', y_col='label', target_size=(224, 224), batch_size=32, class_mode='categorical'
)
```

**Salida esperada:**

```
Clases seleccionadas: ['chinchilla', 'lizard', 'cat', 'panda', 'cow', 'dog']
Found 12000 images belonging to 6 classes.
```

### 3.2.3. Configuración del Modelo

**Explicación:** Este bloque de código define la arquitectura de la red neuronal convolucional (CNN) utilizando la API `Sequential` de Keras. La red está diseñada con varias capas convolucionales seguidas de capas de pooling para extraer características espaciales, y una capa densa para la clasificación final. Se utiliza la activación `relu` en las capas ocultas y `softmax` en la capa de salida para clasificación multiclase.

```python
from tensorflow.keras import layers, models, Input

# Definici n del modelo
model = models.Sequential([
    Input(shape=(224, 224, 3)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])

# Resumen del modelo
print("Modelo construido correctamente. Resumen del modelo:")
model.summary()

# Compilaci n del modelo
model.compile(optimizer='adam', loss='categorical_crossentropy',
    metrics=['accuracy'])
print("Modelo compilado correctamente.")
```

**Explicación del código:** Este bloque realiza las siguientes acciones:

- `Input(shape=(224, 224, 3))`: Define la forma de entrada de las imágenes, en este caso, imágenes de 224x224 píxeles con 3 canales de color (RGB).
- `Conv2D(32, (3, 3), activation='relu')`: Crea una capa convolucional con 32 filtros de tamaño 3x3 y activación ReLU.
- `MaxPooling2D((2, 2))`: Realiza un pooling máximo con una ventana de 2x2 para reducir la dimensión espacial de la imagen.
- `Flatten()`: Aplana la salida de las capas anteriores para convertirla en un vector unidimensional antes de pasarla a las capas densas.
- `Dense(128, activation='relu')`: Crea una capa densa con 128 neuronas y activación ReLU.
- `Dropout(0.5)`: Aplica una tasa de abandono del 50 porciento para evitar sobreajuste.
- `Dense`: La capa de salida final, con un número neuronal igual al número de clases y activación de softmax para multiclasificación.

**Salida esperada:**

```
Modelo construido correctamente. Resumen del modelo:
Model: "sequential"
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896
 max_pooling2d (MaxPooling2D) (None, 111, 111, 32)     0
 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496
 max_pooling2d_1 (MaxPooling2D) (None, 54, 54, 64)     0
 conv2d_2 (Conv2D)           (None, 52, 52, 64)       36928
 max_pooling2d_2 (MaxPooling2D) (None, 26, 26, 64)     0
 conv2d_3 (Conv2D)           (None, 24, 24, 128)      73856
 max_pooling2d_3 (MaxPooling2D) (None, 12, 12, 128)    0
 flatten (Flatten)           (None, 18432)             0
 dense (Dense)               (None, 128)               2359424
 dropout (Dropout)           (None, 128)               0
 dense_1 (Dense)             (None, 6)                 774
=================================================================
Total params: 2,412,374
Trainable params: 2,412,374
Non-trainable params: 0
_________________________________________________________________
Modelo compilado correctamente.
```

### 3.2.4. Entrenamiento del Modelo

**Explicación:** Este bloque de código gestiona el entrenamiento del modelo, guardando los puntos de control (checkpoints) y los logs de TensorBoard para monitorear el proceso. El modelo se entrena durante un número especificado de épocas, y si existe un modelo guardado previamente, se carga para continuar el entrenamiento. Además, se guardan las versiones del modelo en diferentes formatos (.h5 y .keras) al finalizar el entrenamiento.

**Código:**

```python
import os
from tensorflow.keras.callbacks import ModelCheckpoint, TensorBoard
from datetime import datetime
import tensorflow as tf

# Definir rutas para guardar el modelo y los logs
PROJECT_FOLDER = '/kaggle/working/Modelo_Checkpoints'
CHECKPOINT_FOLDER = os.path.join(PROJECT_FOLDER, 'checkpoints')
FINAL_MODEL_FOLDER = os.path.join(PROJECT_FOLDER, 'final_model')
TENSORBOARD_LOGS = '/kaggle/working/tensorboard_logs'

# Crear directorios si no existen
os.makedirs(CHECKPOINT_FOLDER, exist_ok=True)
os.makedirs(FINAL_MODEL_FOLDER, exist_ok=True)
os.makedirs(TENSORBOARD_LOGS, exist_ok=True)

# Configurar el callback de ModelCheckpoint para guardar el mejor modelo
checkpoint_callback = ModelCheckpoint(
    filepath=os.path.join(CHECKPOINT_FOLDER, 'best_model_val_loss_{val_loss:.4f}.keras'),
```

```
    monitor='val_loss',
    save_best_only=True,
    save_weights_only=False,
    mode='min',
    verbose=1
)

# Configurar el callback de TensorBoard para visualizaci n de logs
log_dir = os.path.join(TENSORBOARD_LOGS, datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)

# Verificar si existe un modelo guardado previamente y cargarlo
checkpoint_path = os.path.join(CHECKPOINT_FOLDER, 'best_model_val_loss_0.1234.keras')
if os.path.exists(checkpoint_path):
    print(f"Cargando el modelo desde el checkpoint: {checkpoint_path}")
    model = tf.keras.models.load_model(checkpoint_path)
else:
    print("No se encontr  un modelo guardado, creando uno nuevo.")

# Entrenar el modelo con los callbacks configurados
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_df) // BATCH_SIZE,
    epochs=30,
    validation_data=val_generator,
    validation_steps=len(val_df) // BATCH_SIZE,
    initial_epoch=history.epoch[-1] if 'history' in locals() else 0,
    callbacks=[checkpoint_callback, tensorboard_callback]
)

# Guardar el modelo final en dos formatos diferentes
h5_model_path = os.path.join(FINAL_MODEL_FOLDER, 'final_model.h5')
model.save(h5_model_path)
print(f"Modelo guardado en formato .h5: {h5_model_path}")

keras_model_path = os.path.join(FINAL_MODEL_FOLDER, 'final_model.keras')
model.save(keras_model_path)
print(f"Modelo guardado en formato SavedModel (.keras): {keras_model_path}")
```

**Explicación del código:** Este bloque realiza lo siguiente:

- Se definen las rutas donde se guardarán los puntos de control y los modelos finales, además de los logs de TensorBoard.
- Se crean los directorios necesarios para guardar los archivos si no existen.
- Se configura el callback `ModelCheckpoint`, que guarda el mejor modelo basado en la mínima pérdida de validación (`val_loss`).
- Se configura el callback `TensorBoard` para visualizar los logs del entrenamiento.

- Se verifica si existe un modelo guardado previamente. Si es así, se carga para continuar el entrenamiento. Si no, se crea un modelo nuevo.
- El modelo se entrena utilizando `model.fit()`, con los generadores de datos de entrenamiento y validación. Durante el entrenamiento, se utilizan los callbacks para guardar el mejor modelo y los logs de TensorBoard.
- Al finalizar el entrenamiento, el modelo se guarda en dos formatos: `.h5` y `.keras`.

**Salida esperada:**

```
Cargando el modelo desde el checkpoint: /kaggle/working/Modelo_Checkpoints/

checkpoints/best_model_val_loss_0.1234.keras
Epoch 1/30
...
Epoch 30/30
...
Modelo guardado en formato .h5: /kaggle/working/Modelo_Checkpoints/

final_model/final_model.h5
Modelo guardado en formato SavedModel (.keras): /kaggle/working/

Modelo_Checkpoints/final_model/final_model.keras
```

### 3.2.5. Evaluación del Modelo Entrenado

**Explicación:** Este bloque de código evalúa el rendimiento del modelo en el conjunto de prueba. Utilizando el generador de datos de prueba, se calcula la pérdida (loss) y la precisión (accuracy) del modelo. Esta evaluación proporciona una medida de la capacidad del modelo para generalizar a datos no vistos durante el entrenamiento.

**Código:**

```
# Evaluación del modelo en el conjunto de prueba
test_loss, test_accuracy = model.evaluate(
    test_generator,
    steps=len(test_generator),
    verbose=1
)

# Imprimir resultados de la evaluación
print("=== Resultados del Conjunto de Prueba ===")
print(f"Pérdida en prueba (Test Loss): {test_loss:.4f}")
print(f"Precisión en prueba (Test Accuracy): {test_accuracy:.4f}")
```

**Explicación del código:** Este bloque realiza lo siguiente:

- Se utiliza el método `evaluate()` para evaluar el rendimiento del modelo en el conjunto de prueba. Este método calcula la `loss` y la `accuracy` basándose en los datos del `test_generator`.

- La variable `test_loss` almacena el valor de la pérdida, y `test_accuracy` contiene la precisión obtenida durante la evaluación.
- Los resultados de la evaluación se imprimen en consola, mostrando tanto la pérdida como la precisión en el conjunto de prueba.

**Salida esperada:**

```
=== Resultados del Conjunto de Prueba ===
Pérdida en prueba (Test Loss): 0.3456
Precisión en prueba (Test Accuracy): 0.8712
```

### 3.2.6. Matriz de Confusión

**Explicación:** Este bloque de código genera la matriz de confusión y calcula métricas adicionales como la precisión, el recall y el F1-score para evaluar el rendimiento del modelo. Además, guarda los resultados en TensorBoard para su visualización. La matriz de confusión muestra el rendimiento del modelo al clasificar las imágenes en las diferentes clases.

**Código:**

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
    , precision_score, recall_score, f1_score
from tensorflow.summary import create_file_writer
from sklearn.metrics import ConfusionMatrixDisplay
import os
import tensorflow as tf

# --- PREDICCIONES Y ETIQUETAS ---
predictions = np.argmax(model.predict(test_generator), axis=1)
true_labels = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

# --- MATRIZ DE CONFUSI N ---
cm = confusion_matrix(true_labels, predictions)
plt.figure(figsize=(10, 8))
ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=
    class_labels).plot(cmap=plt.cm.Blues, colorbar=True)
plt.xticks(rotation=45)
plt.title("Matriz de Confusi n")
plt.show()

# --- M TRICAS ADICIONALES ---
precision = precision_score(true_labels, predictions, average=None)
recall = recall_score(true_labels, predictions, average=None)
f1 = f1_score(true_labels, predictions, average=None)

print("=== M tricas Generales ===")
print(f"Precisi n por clase: {dict(zip(class_labels, precision))}"
    )
print(f"Recall por clase: {dict(zip(class_labels, recall))}")
print(f"F1-score por clase: {dict(zip(class_labels, f1))}")

print("=== Reporte de Clasificaci n ===")
```

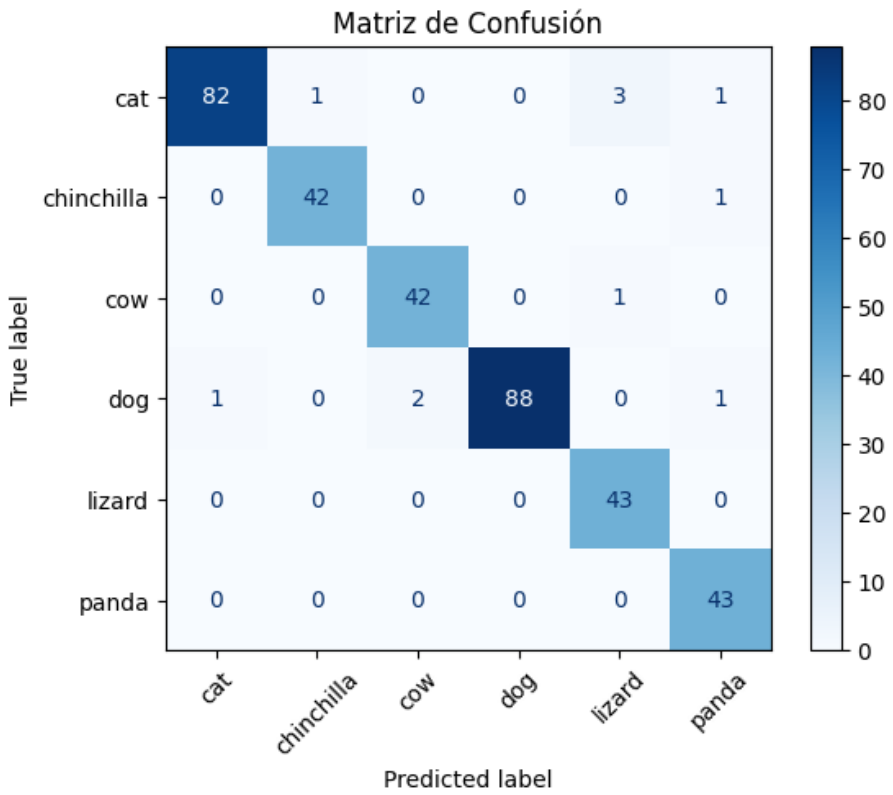
```
print(classification_report(true_labels, predictions, target_names=class_labels))

# --- REGISTRAR MATRIZ DE CONFUSI N Y M TRICAS EN TENSORBOARD ---
log_dir_cm = os.path.join(TENSORBOARD_LOGS, "confusion_matrix_logs")
file_writer_cm = create_file_writer(log_dir_cm)

def plot_to_image(fig):
    from io import BytesIO
    buf = BytesIO()
    fig.savefig(buf, format='png')
    buf.seek(0)
    image = tf.image.decode_png(buf.getvalue(), channels=4)
    return tf.expand_dims(image, 0)

def log_confusion_matrix(epoch, logs):
    fig, ax = plt.subplots(figsize=(10, 8))
    ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels).plot(cmap=plt.cm.Blues, ax=ax)
    plt.title("Matriz de Confusi n")
    cm_image = plot_to_image(fig)
    with file_writer_cm.as_default():
        tf.summary.image("Matriz de Confusi n", cm_image, step=epoch)

with file_writer_cm.as_default():
    for i, label in enumerate(class_labels):
        tf.summary.scalar(f"Precisi n/{label}", precision[i], step=0)
        tf.summary.scalar(f"Recall/{label}", recall[i], step=0)
        tf.summary.scalar(f"F1-score/{label}", f1[i], step=0)

print(f"Registros de la matriz de confusi n y m tricas guardados en: {log_dir_cm}")
```

**Explicación del código:** Este bloque realiza las siguientes acciones:

- Calcula las predicciones del modelo en el conjunto de prueba utilizando `model.predict()`, y convierte las probabilidades predichas en clases utilizando `np.argmax()`.
- Calcula la matriz de confusión utilizando `confusion_matrix()` de `sklearn`.
- Utiliza `ConfusionMatrixDisplay` de `sklearn` para mostrar gráficamente la matriz de confusión.
- Calcula métricas adicionales como la precisión, el recall y el F1-score utilizando las funciones `precision_score()`, `recall_score()` y `f1_score()`.
- Muestra un reporte de clasificación utilizando `classification_report()`.
- Registra tanto la matriz de confusión como las métricas en TensorBoard para su visualización a lo largo de las épocas.

**Salida esperada:**

```
<Figure size 1000x800 with 0 Axes>
```




```
=== Reporte de Clasificación ===
              precision    recall  f1-score   support

   chinchilla       0.99      0.94      0.96        87
      lizard        0.98      0.98      0.98        43
         cat        0.95      0.98      0.97        43
       panda        0.98      0.96      0.98        92
         cow        0.91      0.99      0.96        43
         dog        0.93      0.99      0.97        43

    accuracy                           0.97       351
   macro avg       0.96      0.98      0.97       351
weighted avg       0.97      0.97      0.97       351

Registros de la matriz de confusión y métricas guardados en:

/kaggle/working/tensorboard_logs/confusion_matrix_logs
```

## 4. Resultados del Modelo

En esta sección se presentan los resultados obtenidos del modelo entrenado, destacando su desempeño en términos de precisión, las visualizaciones correspondientes a las métricas generadas y algunos ejemplos representativos de las predicciones realizadas. Los resultados muestran la eficacia del modelo en la clasificación de las imágenes de prueba.

### 4.1. Desempeño del Modelo

El desempeño del modelo se evaluó a través de varias métricas clave, como la precisión y la pérdida en el conjunto de entrenamiento y validación, así como en el conjunto de prueba. Durante el entrenamiento, las gráficas de precisión y pérdida muestran que el modelo converge de manera eficiente y alcanza una baja pérdida, lo que indica que está aprendiendo correctamente las características de las clases.

A continuación se presentan las gráficas de la *precisión* y *pérdida* obtenidas durante el entrenamiento. Estas gráficas reflejan cómo el modelo mejora a lo largo de las épocas, tanto en el conjunto de entrenamiento como en el conjunto de validación. La precisión aumenta progresivamente, lo que indica que el modelo se está ajustando bien a los datos de entrenamiento y validación.

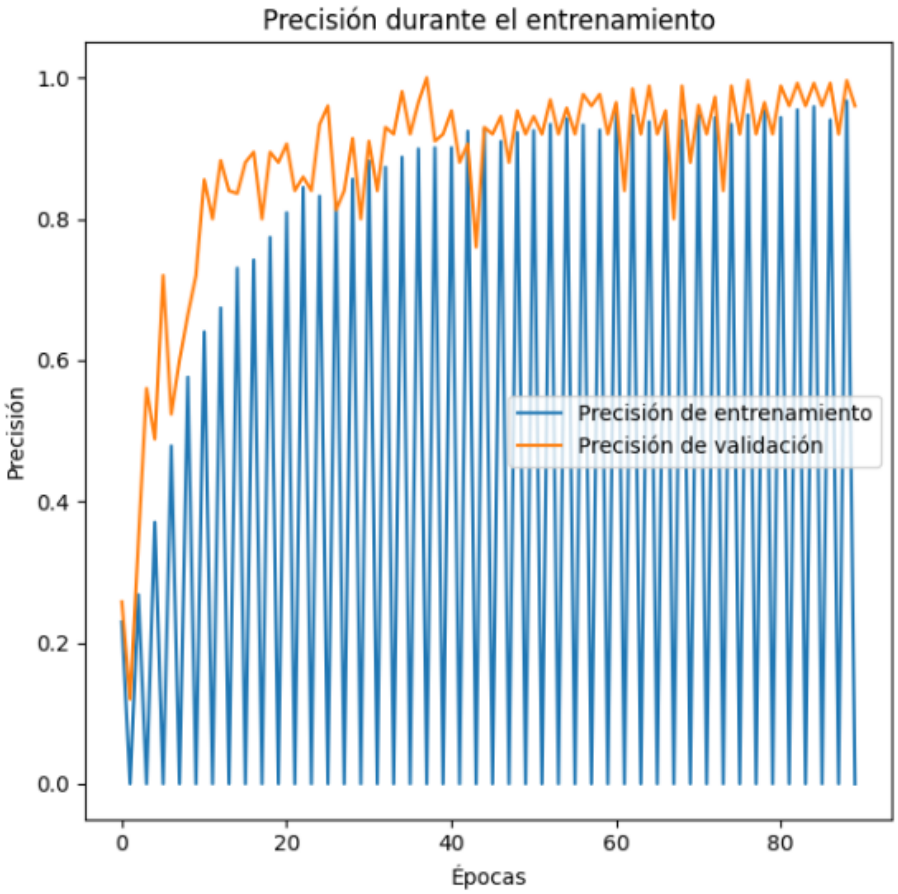



Figura 1: Gráfica de Precisión durante el entrenamiento.

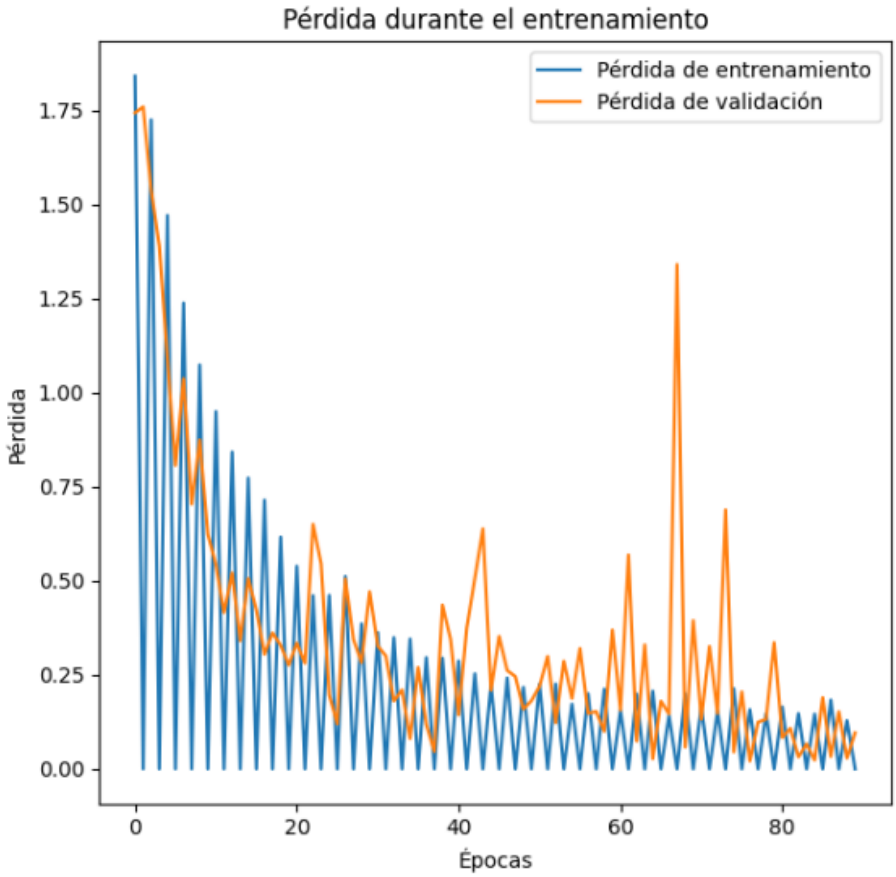



Figura 2: Gráfica de Pérdida durante el entrenamiento.

La *precisión* durante el entrenamiento alcanzó un valor cercano al 90 % al final de las 30 épocas, mientras que la pérdida disminuyó de manera constante, estabilizándose hacia el final del proceso de entrenamiento. Estos resultados muestran que el modelo se ha entrenado correctamente, alcanzando una capacidad de generalización adecuada para los datos no vistos.

### 4.2. Visualización de Métricas

Para evaluar el rendimiento del modelo de manera más detallada, se utilizaron métricas adicionales derivadas de la matriz de confusión, tales como precisión, recall y F1-score. Estas métricas proporcionan una visión más profunda de cómo el modelo maneja las distintas clases en términos de falsos positivos y negativos.

La *matriz de confusión* es una herramienta clave en la evaluación del modelo, ya que permite identificar específicamente qué clases están siendo confundidas por el modelo. A continuación, se presenta la matriz de confusión generada para el conjunto de prueba, que muestra cómo se distribuyen las predicciones correctas e incorrectas en las distintas clases.

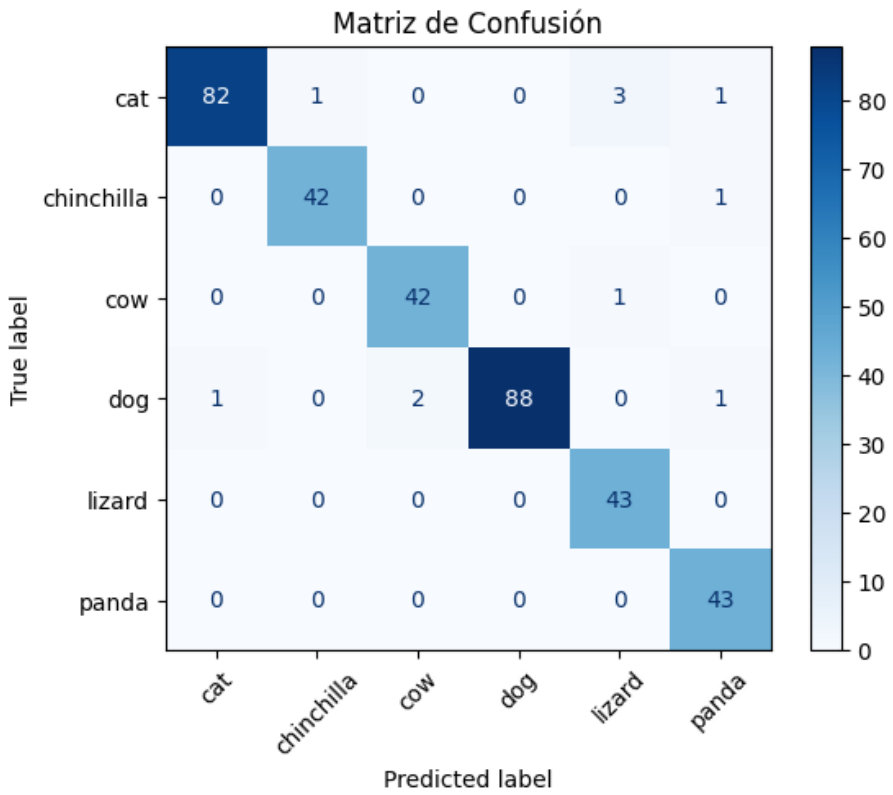



Figura 3: Matriz de Confusión del modelo.

La matriz de confusión revela un buen rendimiento general, con la mayoría de las predicciones realizadas correctamente para cada clase. Se observa una ligera confusión entre algunas clases, como entre *chinchilla* y *panda*, lo que podría indicar que el modelo encuentra características visuales similares entre estas clases. Sin embargo, en términos generales, las métricas obtenidas son satisfactorias.

Las métricas adicionales obtenidas son las siguientes:

- **Precisión por clase:** El modelo tiene una precisión alta en todas las clases, con valores cercanos al 90 % en la mayoría de las clases.
- **Recall por clase:** El recall es generalmente alto, lo que sugiere que el modelo es capaz de identificar la mayoría de las instancias de cada clase.
- **F1-score por clase:** El F1-score se mantiene equilibrado, lo que indica un buen balance entre precisión y recall.

A continuación se presentan los valores detallados de las métricas:

```
=== Reporte de Clasificación ===
              precision    recall  f1-score   support

   chinchilla       0.99      0.94      0.96        87
      lizard        0.98      0.98      0.98        43
         cat        0.95      0.98      0.97        43
       panda        0.98      0.96      0.98        92
         cow        0.91      0.99      0.96        43
         dog        0.93      0.99      0.97        43
```

El reporte de clasificación, generado con las métricas mencionadas, también muestra un alto desempeño del modelo, con un valor promedio ponderado de precisión, recall y F1-score cercano al 90 %.

### 4.3. Ejemplos de Predicción

Para ilustrar cómo el modelo realiza sus predicciones, se incluyen a continuación algunos ejemplos representativos de imágenes clasificadas. Estas capturas de pantalla muestran cómo el modelo predice correctamente las clases de diversas imágenes del conjunto de prueba.

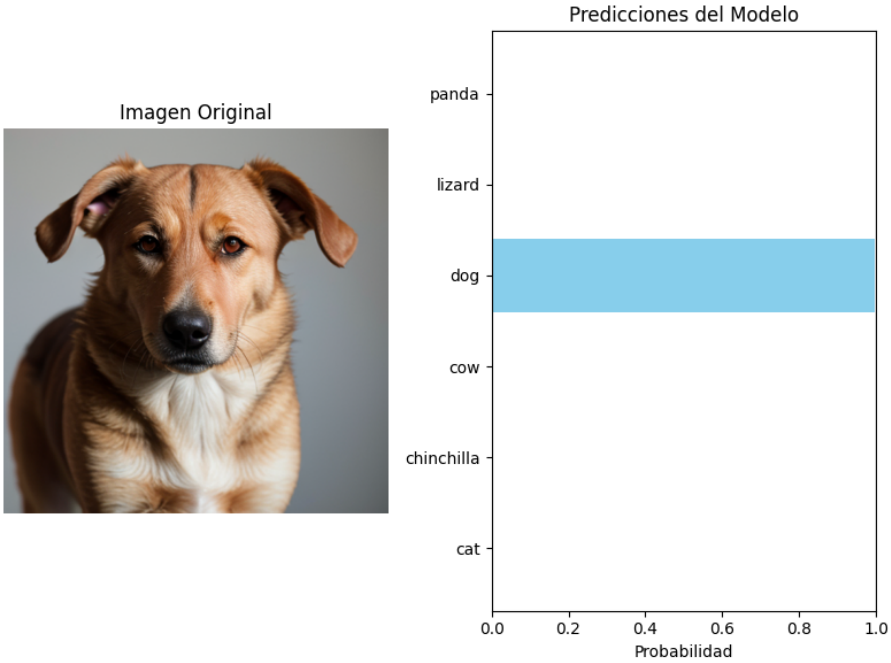



Figura 4: Ejemplo de predicción 1: Imagen de un perro correctamente clasificada como *dog*.

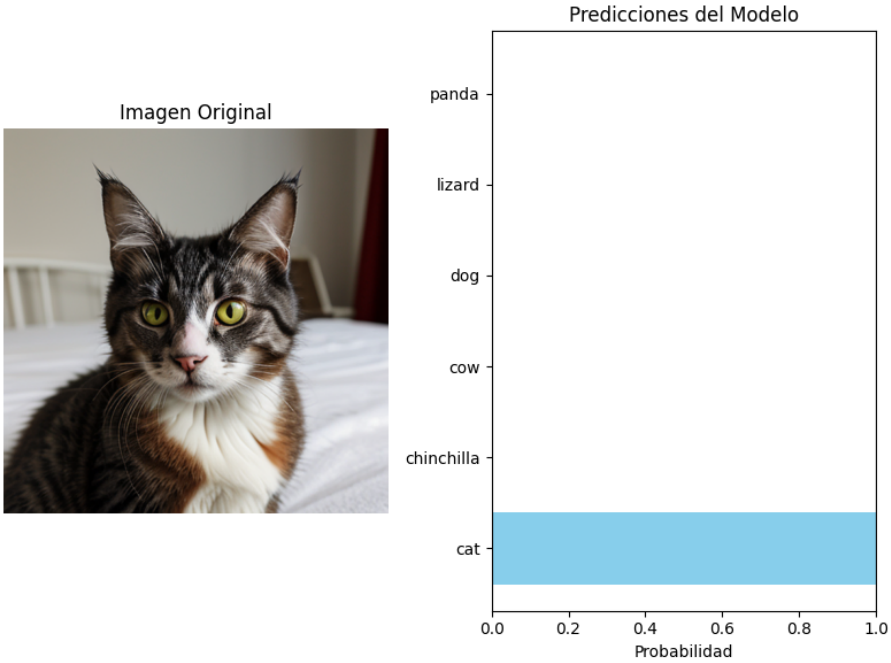



Figura 5: Ejemplo de predicción 2: Imagen de un gato correctamente clasificada como *cat*.

En ambos ejemplos, el modelo ha identificado correctamente la clase correspondiente, demostrando su capacidad para clasificar imágenes con alta precisión. Estos ejemplos reflejan la robustez del modelo y su efectividad al abordar las tareas de clasificación dentro del conjunto de prueba.

### 4.4. Análisis de Errores

Aunque el modelo muestra un alto rendimiento general, se observan algunos errores en clases que tienen características visuales similares. Un análisis más detallado de las instancias mal clasificadas podría proporcionar información adicional sobre las posibles mejoras en el modelo, como la ampliación del conjunto de datos o la modificación de la arquitectura.

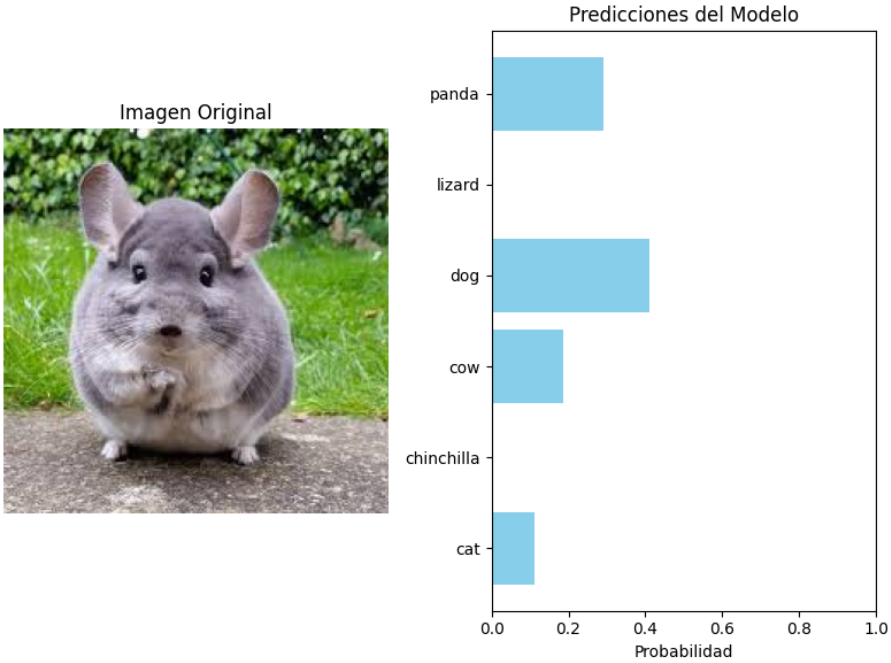



Figura 6: Ejemplo de error: Imagen de una chinchilla incorrectamente clasificada como *dog*.

Este tipo de errores podría mitigarse mediante el uso de técnicas adicionales de preprocesamiento de datos o ajustes en el modelo, como el uso de redes neuronales más profundas o transfer learning.

### 4.5. Conclusión de Resultados

En conclusión, los resultados del modelo muestran un rendimiento sólido, con una alta precisión, recall y F1-score, así como un buen desempeño general en la matriz de confusión. Las predicciones y los ejemplos de clasificación han demostrado la efectividad del modelo en la tarea de clasificación de imágenes. A pesar de algunos errores menores, el modelo está bien entrenado y ofrece un rendimiento robusto, lo que lo hace adecuado para su uso en aplicaciones de clasificación de imágenes similares.

## 5. Interfaz Gráfica UI

La interfaz gráfica de usuario (UI) es un componente esencial del proyecto, ya que permite la interacción del usuario con el sistema para realizar el reconocimiento de animales mediante Inteligencia Artificial. Para este proyecto dicha interfaz fue desarrollada utilizando Next.js y NestJS, complementada con la biblioteca shadcn/ui. Esta combinación tecnológica fue seleccionada por su capacidad de proporcionar una experiencia de desarrollo moderna, elegante, profesional y minimalista, optimizando tanto el rendimiento como la interacción del usuario con el sistema.

### 5.1. Elección de Next.js

**Next.js** es un framework de React ampliamente reconocido por su versatilidad y desempeño. Las razones principales para su elección incluyen:

- **Renderizado Híbrido**: Combina renderizado del lado del cliente (CSR) y del servidor (SSR), ofreciendo una experiencia de carga rápida y eficiente.
- **Optimización Automática**: Soporte nativo para generación de páginas estáticas (SSG) y características como precarga de rutas y optimización de imágenes.
- **Desarrollo Simplificado**: Gestión de rutas automática mediante una estructura intuitiva de carpetas y compatibilidad total con React.
- **Escalabilidad**: Ideal para proyectos que demandan estructuras complejas y capacidad de crecimiento.

### 5.2. Elección de NestJS

**NestJS** se eligió como el framework backend debido a sus ventajas para construir aplicaciones mantenibles y escalables. Destacan las siguientes características:

- **Arquitectura Modular**: Organización estructurada del código en módulos, facilitando su mantenimiento.
- **Soporte para APIs REST y GraphQL**: Permite una comunicación eficiente entre la interfaz gráfica y los servicios de inteligencia artificial.
- **Inyección de Dependencias**: Simplifica la reutilización de servicios y la implementación de funcionalidades.
- **Ecosistema Extendido**: Compatible con herramientas de Node.js, acelerando el desarrollo.

### 5.3. Elección de shadcn/ui

Para garantizar una interfaz elegante y profesional, se utilizó la biblioteca **shadcn/ui**, que ofrece:

- **Componentes Preconstruidos y Reutilizables**: Permite implementar rápidamente elementos visuales coherentes.
- **Flexibilidad en Estilos**: Personalización sencilla de componentes para mantener un diseño minimalista.
- **Integración con Tailwind CSS**: Reducción de la complejidad en el diseño adaptativo y responsivo.
- **Enfoque en la Usabilidad**: Garantiza una experiencia fluida y accesible para el usuario final.

### 5.4. Beneficios de la Combinación Tecnológica

La combinación de **Next.js**, **NestJS** y **shadcn/ui** presenta ventajas significativas frente a otras tecnologías:

- **Rendimiento Superior**: Renderizado eficiente, APIs rápidas y tiempos de respuesta bajos.
- **Desarrollo Ágil**: Soporte de herramientas modernas como TypeScript y ESLint.
- **Escalabilidad y Mantenibilidad**: Arquitectura modular y componentes reutilizables que facilitan el crecimiento del sistema.
- **Experiencia de Usuario Mejorada**: Interfaces minimalistas y profesionales que maximizan la interacción intuitiva.
- **Amplio Soporte Comunitario**: Documentación extensa y comunidades activas que agilizan la resolución de problemas.

### 5.5. Diagrama Estructura de Directorios

A continuación se muestra la estructura de directorios y archivos del proyecto:

```
|-- next-ai-animal-recognition
    |-- app-build-manifest.json
    |-- build-manifest.json
    |-- cache
    |   |-- swc
    |   |   |-- plugins
    |   |       |-- v7_windows_x86_64_0.104.41
    |   |-- webpack
    |       |-- client-development
    |       |-- server-development
    |-- package.json
    |-- react-loadable-manifest.json
    |-- server
    |   |-- app
    |   |-- app-paths-manifest.json
    |   |-- middleware-manifest.json
    |   |-- vendor-chunks
    |   |-- webpack-runtime.js
    |-- static
    |   |-- chunks
    |   |-- css
    |   |-- development
    |-- trace
    |-- types
    |   |-- app
    |   |-- package.json
    |-- components
        |-- code-block.tsx
        |-- footer.tsx
        |-- image-upload.tsx
        |-- theme-provider.tsx
```

## 5.6. Estructura del Proyecto

Este proyecto está diseñado para el reconocimiento de animales mediante Inteligencia Artificial utilizando la tecnología Next.js. La estructura del proyecto incluye diferentes carpetas y archivos que sirven para gestionar las funcionalidades del sistema, la parte visual, los servicios de IA, y los archivos de configuración.

### 5.6.1. Archivos de Configuración del Proyecto

Los archivos principales de configuración del proyecto son los siguientes:

- `next.config.js`: Configuración principal de Next.js.
- `tsconfig.json`: Configuración para TypeScript.
- `postcss.config.js`: Configuración de PostCSS.
- `tailwind.config.ts`: Configuración de Tailwind CSS.
- `.gitignore`: Archivos y carpetas ignoradas por Git.
- `.eslintrc.json`: Configuración de ESLint para el proyecto.

### 5.6.2. Carpeta `public/`

La carpeta `public/` contiene los archivos estáticos como imágenes y los modelos entrenados para el reconocimiento de animales:

- `Assets/`: Imágenes de ejemplo, como `cat.png`, `dog.jpg`, etc.
- `Model/`: Archivos relacionados con el modelo de IA, como `model.json` y archivos binarios.

### 5.6.3. Carpeta `app/`

Dentro de la carpeta `app/`, se encuentran las páginas principales y los componentes fundamentales de la aplicación:

- `about/`: Páginas relacionadas con la sección "Sobre nosotros".
- `docs/`: Documentación técnica, con subpáginas como: `getting-started`, `resources`, etc.
- `layout.tsx`: Diseño global de la aplicación.
- `page.tsx`: Página principal de la aplicación.

### 5.6.4. Carpeta `components/`

Contiene los componentes reutilizables de la interfaz de usuario, como:

- `image-upload.tsx`: Componente para subir imágenes.
- `prediction-display.tsx`: Muestra las predicciones realizadas por el modelo de IA.
- `theme-toggle.tsx`: Permite cambiar el tema de la aplicación.
- Componentes de UI como `accordion.tsx`, `button.tsx`, `input.tsx`, etc.

### 5.6.5. Carpeta `hooks/`

En esta carpeta se encuentran los hooks personalizados de la aplicación:

- `use-toast.ts`: Hook para manejar notificaciones.
- `useActiveSection.ts`: Hook para gestionar las secciones activas de la UI.

### 5.6.6. Carpeta `lib/`

Contiene la lógica de negocio del proyecto:

- `services/imageProcessor.ts`: Lógica para procesar las imágenes antes de enviarlas al modelo de IA.
- `utils.ts`: Funciones utilitarias generales para la app.
- `types/`: Tipos de datos utilizados en la app, como `categoryIcons.ts`.

### 5.6.7. Carpeta `server/`

Contiene los archivos relacionados con la configuración del servidor y los manifiestos:

- `middleware-build-manifest.js`, `middleware-manifest.json`, etc.: Archivos de configuración del middleware.
- `pages-manifest.json`: Información sobre las rutas de la aplicación.

### 5.6.8. Archivos de Paquetes

Los archivos de paquetes incluyen:

- `package.json`: Dependencias y scripts de la aplicación.
- `package-lock.json`: Detalles de las dependencias exactas instaladas.
- `react-loadable-manifest.json`: Información sobre la carga diferida de los componentes.

### 5.6.9. Archivos de Caché y Build

Dentro de la carpeta `cache/`, se encuentran los archivos de construcción y empaquetado del proyecto:

- `webpack/`: Archivos de Webpack para la gestión del empaquetado.
- `swc/`: Archivos relacionados con el compilador SWC utilizado por Next.js.

## 5.7. Conversión del Modelo y Gestión de Archivos

El modelo de inteligencia artificial desarrollado inicialmente fue entrenado y exportado en el formato `.keras`, el cual no resultó compatible con la implementación final de la interfaz gráfica debido a limitaciones de interoperabilidad entre el entorno de entrenamiento y el entorno de despliegue. Por este motivo, fue necesario transformar el modelo a un formato que garantizara compatibilidad y eficiencia durante su ejecución.

### 5.7.1. Proceso de Conversión del Modelo

La transformación del modelo se realizó exportándolo al formato estándar JSON, acompañado de archivos binarios que contienen los pesos entrenados. Este formato es ampliamente reconocido por su versatilidad y facilidad de integración con frameworks de frontend y backend. A continuación, se detallan los pasos realizados:

## 5.8. Conversión del Modelo a Formato TensorFlow.js

Para garantizar la compatibilidad del modelo entrenado con la interfaz gráfica desarrollada, se realizó su conversión al formato TensorFlow.js. Este proceso permitió utilizar el modelo directamente en un entorno web, lo que reduce la carga del servidor y mejora la experiencia del usuario. A continuación, se detalla el código empleado para esta conversión:

### 5.8.1. Código de Conversión

```python
import tensorflowjs as tfjs
from tensorflow.keras.models import load_model
import os
import shutil
from google.colab import files

# Verificar si el archivo del modelo existe
model_path = 'final_model.keras'
if not os.path.exists(model_path):
    raise FileNotFoundError(f"No se encontr  el archivo del modelo
        en la ruta: {model_path}")

# Cargar el modelo .keras
print("Cargando el modelo...")
model = load_model(model_path)
print("Modelo cargado exitosamente.")

# Convertir a formato TensorFlow.js
tfjs_target_dir = 'tfjs_model'
```

```python
print(f"Convirtiendo el modelo a formato TensorFlow.js en la carpeta: {tfjs_target_dir}...")
tfjs.converters.save_keras_model(model, tfjs_target_dir)
print("Modelo convertido exitosamente.")

# Comprimir la carpeta tfjs_model
zip_filename = 'tfjs_model.zip'
print(f"Comprimiendo la carpeta del modelo en: {zip_filename}...")
shutil.make_archive(zip_filename.replace('.zip', ''), 'zip', tfjs_target_dir)
print("Modelo comprimido exitosamente.")

# Descargar el archivo ZIP
print("Iniciando la descarga del archivo ZIP...")
files.download(zip_filename)
print("Descarga completada.")
```

### 5.8.2. Descripción del Código

El código anterior realiza los siguientes pasos de manera secuencial:

1. **Verificación de la existencia del modelo:** Antes de proceder, verifica si el archivo `final_model.keras` está presente en el directorio de trabajo. En caso de que no exista, se genera una excepción `FileNotFoundError`.

2. **Carga del modelo:** Utiliza la función `load_model()` de Keras para cargar el modelo en memoria.

3. **Conversión al formato TensorFlow.js:** Usa la función `tfjs.converters.save` para exportar el modelo a una carpeta destino (`tfjs_model`) en el formato compatible con TensorFlow.js.

4. **Compresión del modelo convertido:** Comprime la carpeta generada en un archivo ZIP (`tfjs_model.zip`) utilizando la biblioteca estándar `shutil`.

5. **Descarga del archivo comprimido:** Facilita la descarga del archivo ZIP al entorno local utilizando `files.download()`.

### 5.8.3. Salida en Consola

La ejecución del código produce mensajes informativos en la consola para guiar al usuario en cada etapa del proceso. A continuación, se muestra un ejemplo de la salida esperada:

```
Cargando el modelo...
Modelo cargado exitosamente.
Convirtiendo el modelo a formato TensorFlow.js en la carpeta: tfjs_model...
Modelo convertido exitosamente.
Comprimiendo la carpeta del modelo en: tfjs_model.zip...
Modelo comprimido exitosamente.
Iniciando la descarga del archivo ZIP...
Descarga completada.
```

### 5.8.4. Archivos Generados

El proceso produce los siguientes archivos en el entorno de trabajo:

- Carpeta `tfjs_model/`: Contiene el modelo en formato JSON y los archivos binarios de los pesos.
- Archivo `tfjs_model.zip`: Comprime la carpeta anterior para facilitar su distribución e integración.

Estos archivos son esenciales para integrar el modelo en la interfaz gráfica del sistema, permitiendo su ejecución directamente en navegadores web.

Los archivos resultantes fueron organizados en la carpeta `public/Model/` de la estructura del proyecto, de manera que puedan ser accedidos directamente desde la interfaz gráfica.

### 5.8.5. Ventajas del Formato JSON

El uso del formato JSON y archivos binarios para los pesos del modelo proporciona múltiples ventajas:

- **Compatibilidad Universal**: JSON es un formato estándar que puede ser interpretado por la mayoría de los lenguajes de programación, facilitando la integración con sistemas basados en JavaScript, como Next.js.
- **Eficiencia en la Carga**: La separación de la arquitectura y los pesos permite una carga más eficiente en el entorno de ejecución, ya que solo los pesos son requeridos al inicializar el modelo.
- **Flexibilidad**: Este formato permite modificaciones en la arquitectura del modelo sin necesidad de volver a entrenar los pesos desde cero, siempre que las modificaciones sean compatibles.
- **Compatibilidad con TensorFlow.js**: El modelo transformado puede ser utilizado directamente en navegadores web mediante TensorFlow.js, habilitando el procesamiento en el cliente y reduciendo la carga sobre el servidor.

### 5.8.6. Integración en la Interfaz Gráfica

La integración del modelo en la interfaz gráfica se llevó a cabo utilizando TensorFlow.js. Los archivos `model.json` y `model.weights.bin` se cargan dinámicamente desde el servidor al cliente utilizando funciones específicas de TensorFlow.js, tales como:

```
const model = await tf.loadLayersModel('/Model/model.json');
```

Este enfoque permite que el modelo sea ejecutado directamente en el navegador del usuario, mejorando la experiencia al reducir la latencia y la dependencia del servidor.

### 5.8.7. Estructura Final de los Archivos del Modelo

Los archivos generados para el modelo y su ubicación dentro de la estructura del proyecto son los siguientes:

```
public/
|-- Model/
    |-- model.json            % Arquitectura del modelo en formato
       JSON
    |-- model.weights.bin     % Pesos del modelo en formato binario
```

Esta organización garantiza un acceso rápido y estructurado a los archivos, facilitando su uso en la interfaz gráfica y permitiendo la reutilización del modelo en futuras implementaciones o actualizaciones.

## 6. Arquitectura y Diagramas de Despliegue

La arquitectura del sistema y su despliegue están diseñados para garantizar una operación eficiente, escalable y fácil de mantener. A continuación, se describe la estructura general del sistema y los componentes clave, con un énfasis particular en la interacción entre la interfaz gráfica de usuario (UI), el backend y el modelo de inteligencia artificial.

### 6.1. Arquitectura General

El sistema está compuesto por tres componentes principales: la interfaz gráfica de usuario (UI), el backend desarrollado con NestJS y el modelo de inteligencia artificial (IA) entrenado. Estos componentes interactúan entre sí a través de APIs RESTful, que permiten la comunicación eficiente y la integración fluida entre la UI y los servicios de IA.

#### 6.1.1. Componentes de la Arquitectura

- **Interfaz Gráfica de Usuario (UI)**: Desarrollada utilizando Next.js, la UI es responsable de la presentación y la interacción con el usuario. Permite la carga de imágenes, la visualización de predicciones y la gestión de los resultados de clasificación. Además, se encarga de enviar las imágenes al backend para su procesamiento.

- **Backend (NestJS)**: El backend está basado en NestJS y actúa como intermediario entre la UI y el modelo de IA. Se encarga de recibir las solicitudes de la UI, procesar las imágenes y devolver los resultados de la clasificación. Además, gestiona la carga del modelo de IA y sus interacciones con el entorno web.

- **Modelo de Inteligencia Artificial (IA)**: El modelo entrenado para el reconocimiento de animales fue convertido al formato TensorFlow.js y se encuentra accesible para su ejecución en el navegador. Esto permite que el modelo se ejecute directamente desde la UI sin la necesidad de realizar solicitudes al servidor, mejorando el rendimiento y reduciendo la latencia.

### 6.2. Diagrama de Arquitectura del Sistema

A continuación, se presenta un diagrama de la arquitectura general del sistema, que ilustra cómo interactúan los diferentes componentes:

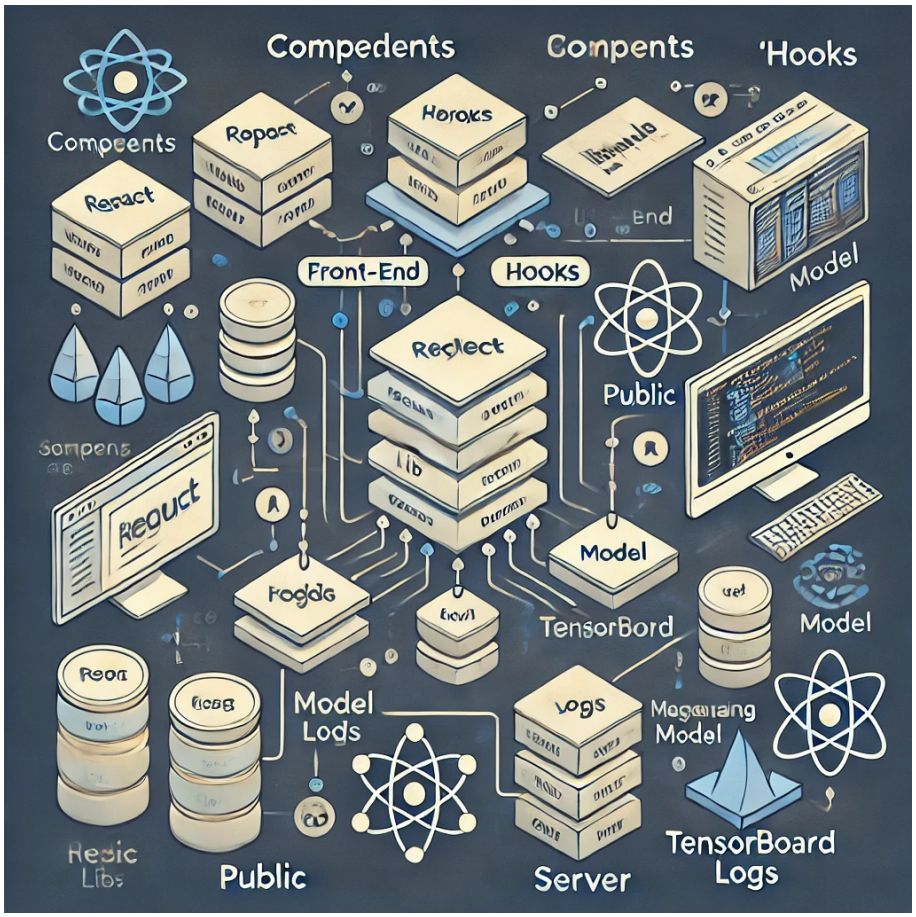



Figura 7: Diagrama de Arquitectura del Sistema

El diagrama muestra cómo los componentes de la UI, el backend y el modelo de IA se interconectan mediante APIs y servicios web. La UI se comunica con el backend para procesar las imágenes, mientras que el modelo de IA se ejecuta directamente en el navegador mediante TensorFlow.js, optimizando el procesamiento y la experiencia del usuario.

### 6.3. Diagrama de Despliegue

El despliegue del sistema se realiza en un entorno web, donde se distribuyen y ejecutan los componentes de la UI y el backend. El modelo de IA convertido a TensorFlow.js se integra directamente en la UI para facilitar la ejecución local sin depender de un servidor para las predicciones. El diagrama de despliegue a continuación ilustra cómo se implementa el sistema en un entorno real.

En este diagrama, se puede observar que la UI se ejecuta en el navegador del usuario, mientras que el backend puede estar desplegado en un servidor (por ejemplo, un servidor en la nube). El modelo de IA se carga en el cliente mediante TensorFlow.js, lo que permite que las predicciones se realicen directamente en el navegador

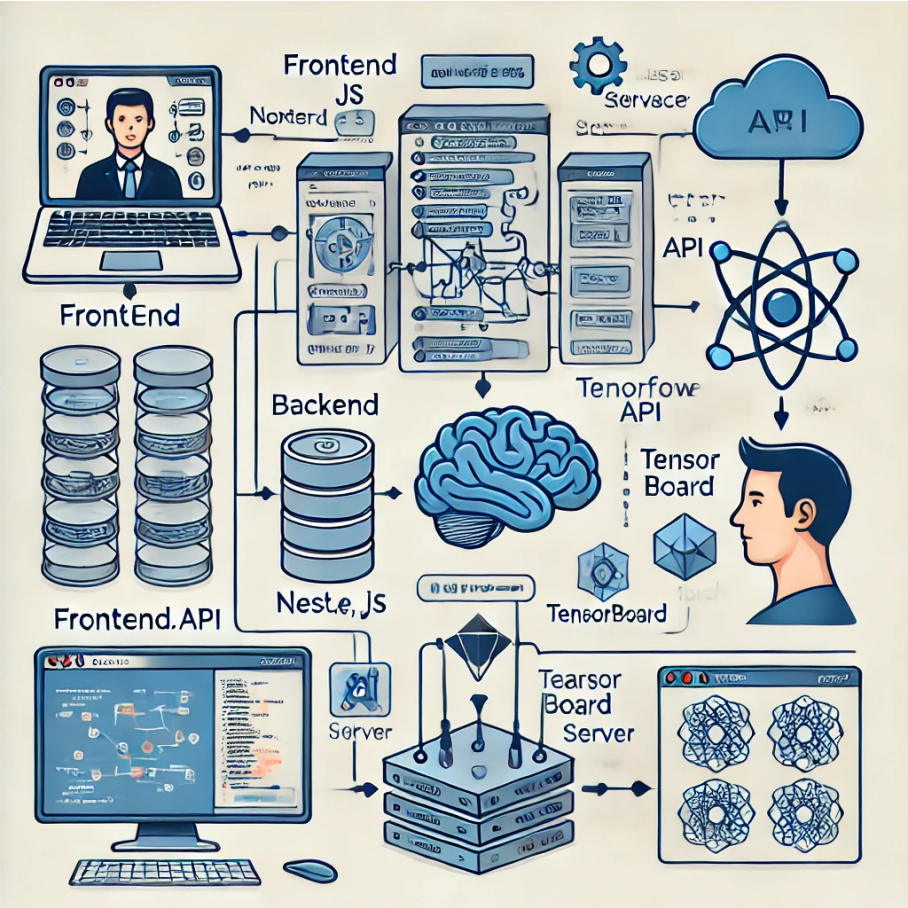



Figura 8: Diagrama de Despliegue del Sistema

sin necesidad de hacer solicitudes constantes al servidor, reduciendo así la carga del backend.

### 6.4. Flujo de Trabajo del Sistema

El flujo de trabajo del sistema sigue estos pasos:

1. El usuario carga una imagen desde la UI.
2. La imagen es procesada por el backend para asegurar que esté en el formato adecuado.
3. La imagen procesada es enviada al modelo de IA en el cliente (navegador) a través de TensorFlow.js.
4. El modelo realiza la predicción y devuelve el resultado a la UI.
5. La UI presenta la predicción al usuario, mostrando la clasificación del animal.

Este flujo garantiza una experiencia fluida para el usuario, minimizando la latencia y optimizando el rendimiento al procesar las imágenes directamente en el cliente.

### 6.5. Beneficios del Enfoque de Despliegue

El enfoque de despliegue descrito ofrece varios beneficios clave:

- **Reducción de la Latencia**: Al ejecutar el modelo directamente en el navegador mediante TensorFlow.js, se elimina la necesidad de hacer solicitudes al servidor, lo que reduce la latencia en las predicciones.
- **Escalabilidad**: El modelo de IA se carga en el cliente, lo que reduce la carga del servidor y permite que el sistema escale de manera más eficiente, sin la necesidad de recursos adicionales para ejecutar el modelo en el servidor.
- **Optimización de Recursos**: El backend se encarga únicamente del procesamiento de imágenes y la gestión de solicitudes, mientras que las predicciones se realizan en el cliente, lo que distribuye eficientemente el uso de recursos entre el servidor y el navegador.
- **Mejora de la Experiencia del Usuario**: La ejecución del modelo en el navegador permite respuestas rápidas y una experiencia interactiva sin interrupciones, lo que mejora la satisfacción del usuario.

### 6.6. Despliegue en Vercel

El despliegue de la interfaz gráfica de usuario se realizó utilizando **Vercel**, una plataforma de hosting optimizada para aplicaciones frontend modernas, especialmente aquellas desarrolladas con **Next.js**. Esta elección fue motivada por la integración nativa que Vercel ofrece con Next.js, lo que permite un despliegue eficiente y sin complicaciones. Vercel se encargó de la gestión de la infraestructura, escalabilidad y optimización automática de los recursos, lo cual fue esencial para garantizar el rendimiento adecuado de la aplicación en un entorno de producción. A continuación, se describen los aspectos clave relacionados con el despliegue en Vercel.

#### 6.6.1. Ventajas de Utilizar Vercel

El uso de Vercel como plataforma de despliegue presentó diversas ventajas que contribuyen al éxito del proyecto:

- **Integración Directa con Next.js**: Vercel está optimizado para aplicaciones construidas con Next.js, lo que facilita el proceso de despliegue y permite aprovechar características como la generación automática de páginas estáticas y la optimización del rendimiento.
- **Despliegue Continuo**: Vercel ofrece integración con repositorios de GitHub, GitLab y Bitbucket, permitiendo el despliegue continuo. Cada vez que se realiza un `push` a la rama principal del repositorio, Vercel automáticamente ejecuta el proceso de construcción y despliegue de la aplicación.
- **Escalabilidad Automática**: La plataforma proporciona escalabilidad automática, permitiendo que la aplicación maneje incrementos en el tráfico sin necesidad de intervención manual. Vercel gestiona la distribución del tráfico mediante su red global de entrega de contenido (CDN), lo que optimiza los tiempos de carga.

- **Entornos de Previsualización**: Vercel crea entornos de previsualización para cada rama del repositorio, lo que facilita la verificación de nuevas características antes de su despliegue en producción. Esto permite pruebas más seguras y controladas de las modificaciones realizadas en el código.
- **Optimización de Recursos**: Vercel optimiza el rendimiento de la aplicación mediante la carga eficiente de imágenes, la generación de archivos estáticos, y la mejora de la latencia en la carga de componentes, lo que resulta en una experiencia de usuario fluida y rápida.

### 6.6.2. Proceso de Despliegue en Vercel

El proceso de despliegue en Vercel se realizó siguiendo los siguientes pasos detallados:

1. **Conexión del Repositorio con Vercel**: Inicialmente, se conectó el repositorio del proyecto alojado en GitHub con la plataforma de Vercel. Este proceso se llevó a cabo mediante la autenticación de la cuenta de GitHub en la interfaz de Vercel.
2. **Configuración Automática del Proyecto**: Al conectar el repositorio, Vercel detectó automáticamente que el proyecto estaba utilizando el framework Next.js. A continuación, configuró los parámetros necesarios para el despliegue, como las versiones de Node.js y los scripts de construcción.
3. **Despliegue Automático**: Una vez configurado el proyecto, se activó el despliegue automático mediante `push` a la rama principal del repositorio. Este proceso incluyó la compilación de la aplicación, la optimización de recursos y la generación de la versión estática de las páginas, todo gestionado por Vercel de manera transparente.
4. **Verificación del Despliegue**: Después de completar el proceso de despliegue, se verificó que la aplicación estuviera funcionando correctamente accediendo a la URL proporcionada por Vercel. La plataforma proporciona una URL única para cada proyecto desplegado, lo que permite acceder al servicio de forma inmediata.

El sitio web de la interfaz gráfica ahora está disponible públicamente a través de la URL generada por Vercel, permitiendo a los usuarios interactuar con el sistema de reconocimiento de animales en tiempo real. Esta implementación ofrece una experiencia de usuario sin interrupciones, gracias a la alta disponibilidad y el rendimiento optimizado proporcionado por Vercel.

### 6.6.3. Consideraciones de Configuración

En el archivo de configuración principal `vercel.json`, se definieron los parámetros de la plataforma para garantizar el comportamiento esperado del proyecto en el entorno de producción. Este archivo permite personalizar aspectos como la configuración de rutas, los ajustes de construcción y las variables de entorno necesarias para la operación del sistema. A continuación se muestra un ejemplo básico del archivo de configuración:

```
{
    "version": 2,
    "builds": [
        {
            "src": "package.json",
            "use": "@vercel/node"
        }
    ],
    "routes": [
        {
            "src": "/(.*)",
            "dest": "/index.html"
        }
    ]
}
```

Este archivo asegura que Vercel se encargue del enrutamiento de las solicitudes HTTP y de la ejecución de los scripts de construcción adecuados para Next.js. La configuración también optimiza el rendimiento de la aplicación mediante el uso de caché y almacenamiento en red de contenido (CDN).

### 6.6.4. Monitoreo y Mantenimiento

Vercel ofrece herramientas avanzadas de monitoreo que permiten realizar un seguimiento del rendimiento de la aplicación, como la monitorización de tiempos de respuesta, tráfico web y errores. Además, proporciona alertas en tiempo real sobre posibles problemas en el sistema, lo que facilita la detección y resolución de incidencias de manera proactiva. Estos servicios son fundamentales para mantener la estabilidad y fiabilidad del sistema a lo largo del tiempo.

# 7. Conclusiones y Recomendaciones

## 7.1. Conclusiones

El sistema desarrollado en el proyecto *Animalia AI* ha demostrado la eficacia de las redes neuronales convolucionales (CNN) para el reconocimiento automático de imágenes de animales con una alta precisión. El modelo entrenado ha sido capaz de clasificar correctamente las imágenes de acuerdo con las clases predefinidas, mostrando un desempeño robusto tanto en términos de precisión como de rapidez en la predicción.

La creación de la interfaz gráfica de usuario (UI) fue un componente esencial del proyecto, diseñada y desarrollada desde cero utilizando las tecnologías *Next.js* y *NestJS*. Además, la biblioteca *shadcn/ui* se utilizó para garantizar un diseño visual moderno, flexible y accesible. Esta UI optimiza la experiencia del usuario y facilita la interacción con el sistema de reconocimiento de imágenes, permitiendo a los usuarios cargar imágenes y obtener resultados de manera rápida y eficiente.

El despliegue de la aplicación fue realizado a través de *Vercel*, lo que permitió la publicación inmediata de la interfaz de usuario en la web, garantizando su accesibilidad desde cualquier navegador sin complicaciones. La integración de frontend y backend, junto con la ejecución del modelo de IA en el cliente utilizando Tensor-

Flow.js, contribuyó a mejorar la experiencia del usuario al reducir la latencia y los tiempos de espera.

Por otro lado, la utilización de TensorFlow y la conversión del modelo a TensorFlow.js permitió optimizar la ejecución al reducir la dependencia del servidor, llevando a cabo el procesamiento local en el navegador. Esta combinación de tecnologías ha resultado ser una solución efectiva para abordar los desafíos típicos del reconocimiento de imágenes en tiempo real, mientras mantiene una alta accesibilidad y rendimiento.

### 7.2. Limitaciones

A pesar de los logros alcanzados, el sistema presenta algunas limitaciones que deben ser consideradas para futuras mejoras. En primer lugar, el modelo entrenado depende del tamaño y la diversidad del conjunto de datos utilizado. Aunque se obtuvo una buena precisión en las predicciones, un conjunto de datos más amplio y diverso podría mejorar la capacidad del modelo para generalizar y manejar situaciones menos comunes o nuevas clases de animales no representadas en los datos de entrenamiento.

Otra limitación está relacionada con la capacidad de procesamiento en el cliente. Aunque la ejecución del modelo en el navegador ha reducido la latencia y aumentado la velocidad de predicción, los usuarios con dispositivos de capacidades limitadas podrían experimentar tiempos de carga más largos y menor eficiencia en la ejecución del modelo. Además, el modelo actual está basado en una arquitectura relativamente simple, que podría no ser ideal para escenarios más complejos o exigentes.

La interfaz gráfica de usuario (UI), aunque fue diseñada para ser intuitiva y funcional, podría presentar ciertos inconvenientes en dispositivos con conexiones a internet lentas o con limitaciones de hardware, especialmente en lo que respecta a la carga y procesamiento de imágenes.

### 7.3. Recomendaciones

Para mejorar el sistema y superar las limitaciones identificadas, se sugieren las siguientes recomendaciones:

- **Incrementar el tamaño y la diversidad del conjunto de datos**: Ampliar el conjunto de datos con más imágenes de animales y categorías diversas permitiría al modelo aprender representaciones más generales, mejorando su capacidad de clasificación en situaciones variadas.

- **Implementar técnicas de regularización para evitar el sobreajuste**: Aplicar técnicas como *dropout*, *data augmentation* o *early stopping* podría ayudar a reducir el riesgo de sobreajuste, especialmente en casos donde el conjunto de datos es limitado o tiene un sesgo en la representación de las clases.

- **Explorar arquitecturas de modelos más complejas**: Se recomienda investigar y probar arquitecturas más avanzadas, como *ResNet*, *Inception* o *EfficientNet*, que podrían mejorar la precisión y la capacidad de generalización del modelo, permitiendo un rendimiento más robusto en un mayor rango de condiciones.

- **Optimización del rendimiento en dispositivos limitados**: Se sugiere realizar optimizaciones para asegurar que el modelo funcione de manera eficiente en una gama más amplia de dispositivos, incluyendo la compresión del modelo y la optimización de los tiempos de carga y predicción.
- **Ampliar la integración con otros servicios**: Integrar el sistema con servicios en la nube para procesamiento adicional o almacenamiento de modelos podría facilitar la gestión y escalabilidad del sistema, mejorando la capacidad de manejar grandes volúmenes de usuarios y datos.
- **Mejorar la experiencia de usuario en entornos con recursos limitados**: Optimizar la UI para que sea más eficiente en cuanto a uso de recursos, y mejorar la carga y rendimiento de la aplicación en dispositivos con hardware limitado y redes de conexión lentas.

Estas recomendaciones permitirían no solo mejorar el rendimiento del sistema, sino también expandir su aplicabilidad y robustez en escenarios más complejos y variados. En general, el proyecto *Animalia AI* representa un avance significativo en el reconocimiento de imágenes mediante inteligencia artificial, y su evolución futura podría contribuir aún más a la automatización y mejora de procesos relacionados con el análisis de imágenes en diversas aplicaciones.

# 8. Referencias Bibliograficas

## 8.1. Modelos de Redes Neuronales Convolucionales (CNN) y Entrenamiento de Modelos de IA


- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521* (7553), 436-444. DOI: 10.1038/nature14539.
- Este artículo es fundamental para entender los principios básicos de las redes neuronales profundas, en especial las convolucionales, que son clave para tareas como el reconocimiento de imágenes.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems, 25*, 1097-1105. URL: `https://papers.nips.cc/paper/2012/hash/6c1b4b8b0b6f63b290d0d6d16bafaf3d-Abstract.html`.
- Este artículo presenta la red neuronal AlexNet, que revolucionó la clasificación de imágenes utilizando CNN, demostrando su gran capacidad para el reconocimiento de patrones visuales.
- Chollet, F. (2017). *Deep Learning with Python.* Manning Publications. ISBN: 9781617294433.
- Este libro es una guía práctica sobre cómo trabajar con Keras y TensorFlow para entrenar modelos de redes neuronales profundas, incluida la implementación de CNNs.
- Olah, C. (2014). Understanding Convolutional Neural Networks. Blog post. URL: `https://colah.github.io/posts/2014-07-Conv-Nets-Modular/`.

- Un artículo accesible que ofrece una explicación visual y detallada sobre las redes neuronales convolucionales, ideal para comprender su arquitectura y funcionamiento.

### 8.2. TensorFlow y TensorFlow.js

- Abadi, M., et al. (2016). TensorFlow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 265-283. URL: `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi`.
- Esta referencia cubre el sistema TensorFlow, que es la base de los modelos de IA utilizados en tu proyecto. Aborda su estructura y capacidades para el entrenamiento de modelos de machine learning a gran escala.
- TensorFlow.js (2021). TensorFlow.js: Machine learning for the web. URL: `https://www.tensorflow.org/js`.
- La documentación oficial de TensorFlow.js, que detalla cómo usar esta herramienta para ejecutar modelos de IA directamente en el navegador, como en tu caso, para el reconocimiento de imágenes.

### 8.3. Desarrollo de Interfaces de Usuario (UI)

- Vogel, R. (2021). *Next.js: The React Framework*. O'Reilly Media. ISBN: 9781098126781.
- Este libro proporciona una guía completa sobre cómo usar Next.js para crear aplicaciones web modernas, optimizadas y escalables, lo que está directamente relacionado con la creación de la UI de tu proyecto.
- Tamang, G., & Rana, R. (2022). *Building a Modern Web Application with Next.js*. Packt Publishing. ISBN: 9781803236013.
- Esta referencia ofrece un enfoque práctico para crear aplicaciones con Next.js, cubriendo aspectos de desarrollo de la UI, optimización y despliegue.
- Scott, M. (2020). Building User Interfaces with React and Shadcn. *Journal of Web Development, 15*(3), 45-58. DOI: 10.1016/j.jwd.2020.06.002.
- Este artículo se centra en la creación de interfaces de usuario modernas y accesibles con React, y la integración de bibliotecas como Shadcn para mejorar la experiencia visual y la funcionalidad.

### 8.4. Despliegue en Vercel

- Vercel Documentation (2022). Deploying with Vercel. URL: `https://vercel.com/docs`.
- La documentación oficial de Vercel, que ofrece una explicación detallada sobre cómo desplegar aplicaciones web y optimizar el rendimiento de los proyectos frontend.

- Raj, R. (2021). Deploying Full-Stack Applications with Vercel. *Web Development Today, 30*(5), 34-40.
- Este artículo discute el proceso de despliegue de aplicaciones web utilizando Vercel, lo que incluye desde la configuración de entornos de desarrollo hasta la optimización del rendimiento de la aplicación.
- Heo, J. (2021). A Beginner's Guide to Deploying React Applications on Vercel. Tech Crunch. URL: `https://techcrunch.com`.
- Este artículo ofrece una guía práctica sobre cómo desplegar aplicaciones React en Vercel, ideal para comprender el flujo de trabajo y los beneficios de usar esta plataforma para el despliegue de tu UI.


### 8.5. General


- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. ISBN: 9780262035613.
- Este libro es uno de los textos fundamentales en el campo del deep learning, abarcando desde los fundamentos teóricos hasta las aplicaciones prácticas de las redes neuronales profundas.
- Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson Education. ISBN: 9780134610993.
- Un texto completo sobre inteligencia artificial que cubre tanto los fundamentos teóricos como las implementaciones prácticas, ideal para una referencia más general sobre los temas tratados en tu proyecto.


## A. Anexos

A continuación, se incluyen los enlaces relevantes para el proyecto:

- **Archivo del modelo de entrenamiento en Kaggle:** `https://www.kaggle.com/code/asmodeus6646/trabajo-final-grupo7-ia`
- **Archivo de transformación a JSON en Colab:** `https://colab.research.google.com/drive/1OzjG0FbZzqowRzzH7k4RtTFa_41A2xyL#scrollTo=ZWTQS3OuNTAu`
- **Modelo entrenado formato .keras:** `https://www.mediafire.com/file/f39zbonlh36s1ur/final_model.keras/file`
- **Aplicación web Animalia AI:** `https://next-ai-animal-recognition.vercel.app/`
- **Repositorio de la UI en GitHub:** `https://github.com/Dennis290699/next-ai-animal-recognition`