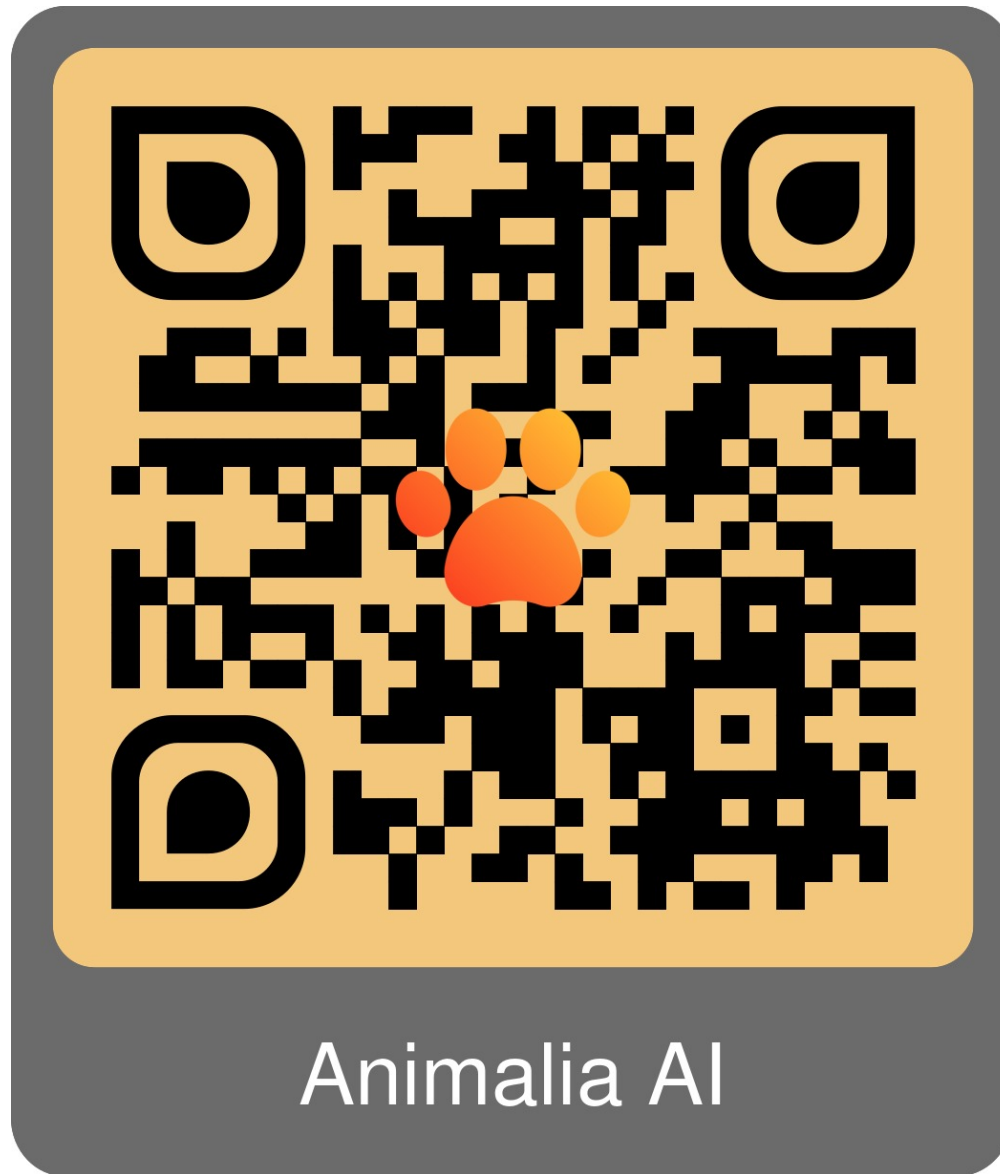


# Proyecto Animalia AI

Reconocimiento de Imágenes con Inteligencia Artificial

Integrantes:

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



Animalia AI



# Introduccion

Desarrollo de un sistema de reconocimiento de imágenes con inteligencia artificial, basado en redes neuronales convolucionales (CNNs) para clasificar especies animales como perro, gato, vaca, lagarto, panda y chinchilla.

# Proceso en fases



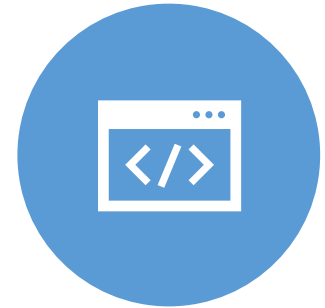
ELECCION DEL  
DATASET



FASE DE  
ENTRENAMIENTO



FASE DE PRUEBAS



FASE DE UI/UX

# Elección del dataset

---

# Recursos



**Plataforma:** Kaggle



**Dataset:** [Image Classification - 64 Classes - Animal](#)

# ¿Qué es Kaggle?



Repositorio de datasets públicos.



Concursos de ciencia de datos con premios.



Herramientas de código y notebooks en la nube.



Comunidad de científicos de datos y desarrolladores de IA.

# Estructura del Dataset



**Numero total de clases: 64**



**Formato de imágenes: 512x512 (PNG)**



**Estructura de carpetas:** Cada clase tiene una carpeta con imágenes del animal correspondiente.



**Clases seleccionadas:** Perro, gato, vaca, lagarto, panda, chinchilla



# Fase de entrenamiento

---

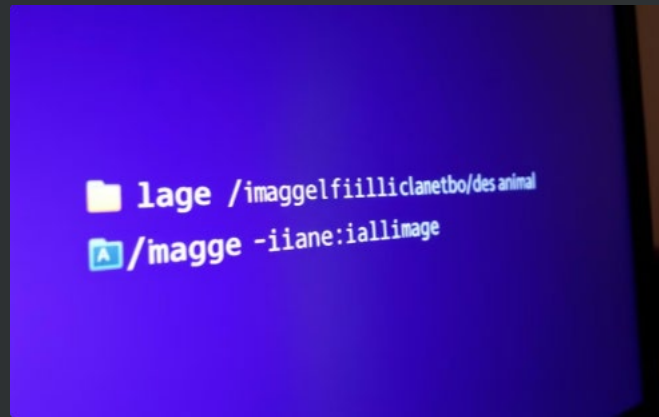
# Fase de Entrenamiento:

## 1. Listar todas las clases disponibles



### Paso 1: Identificar Clases

Identificar las clases de animales en el dataset.



### Paso 2: Ruta del Dataset

Definir la ruta donde se encuentran las imágenes:

```
base_dir = '/kaggle/input/image-  
classification-64-classes-  
animal/image'
```



### Enumeración de Clases

Usar 'os' para listar las clases (carpetas) y mostrarlas con índice numérico.

## 2. Configuraciones iniciales y preparación de datos



### Importación de bibliotecas

Importar bibliotecas necesarias (os, pandas, tensorflow, sklearn).



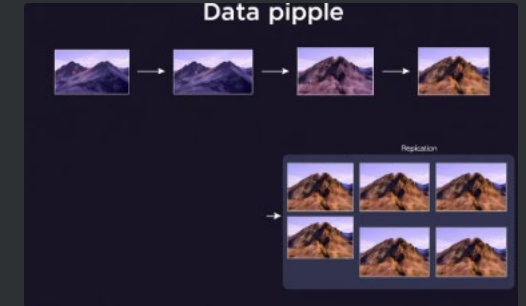
### Selección de clases y carga de datos

Selección de clases (chinchilla, lizard, cat, panda, cow, dog), carga de rutas de imágenes y etiquetas.



### División de datos

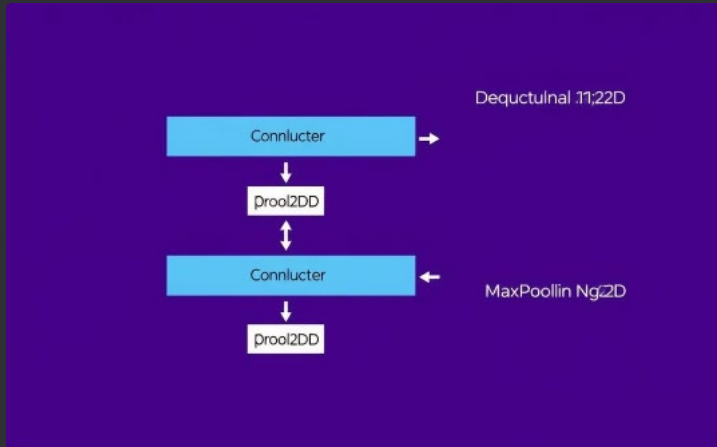
División de datos en conjuntos de entrenamiento, validación y prueba utilizando train\_test\_split.



### Generadores de datos

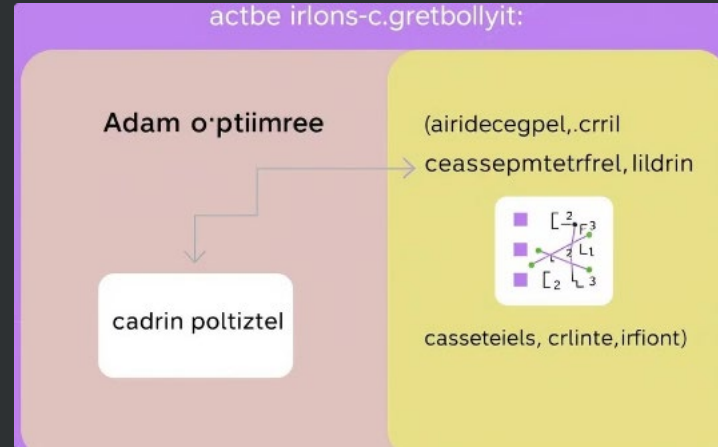
ImageDataGenerator para aumentos de datos y generadores para entrenamiento, validación y prueba.

# 3. Configuración del Modelo



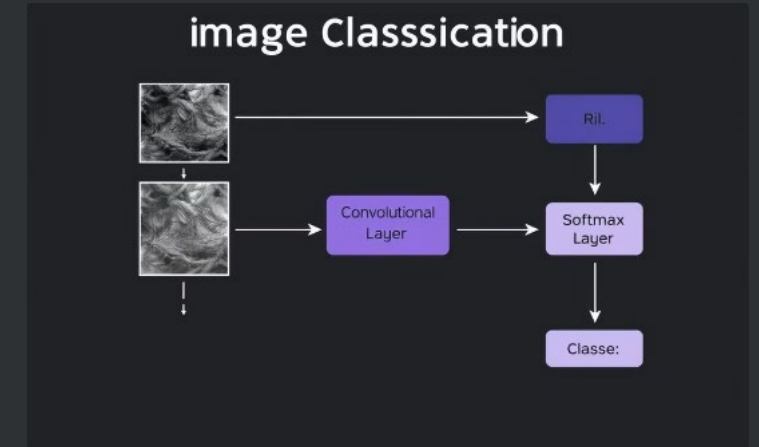
## Construcción del Modelo

Usamos capas convolucionales y de agrupamiento para extraer características de las imágenes, y una capa final para clasificar.



## Compilación

Se utiliza el optimizador Adam y la función de pérdida categorical\_crossentropy.



## Proceso de Clasificación

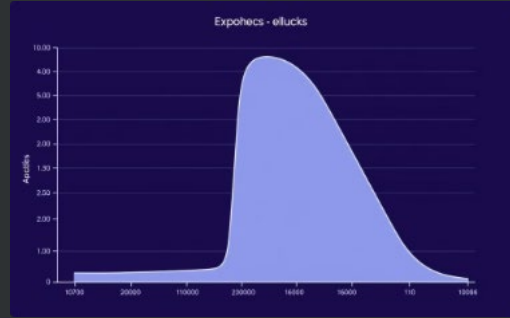
Capas convolucionales extraen características; la capa softmax clasifica en una de las clases.

## 4. Entrenamiento del Modelo



### Configuración de directorios

Creamos carpetas para guardar el progreso.



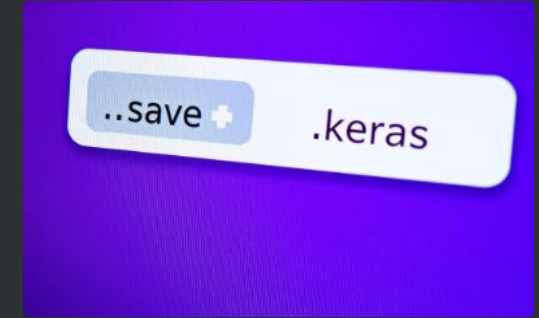
### Callbacks: ModelCheckpoint & TensorBoard

ModelCheckpoint guarda el mejor modelo. TensorBoard registra el entrenamiento para visualización.



### Entrenamiento

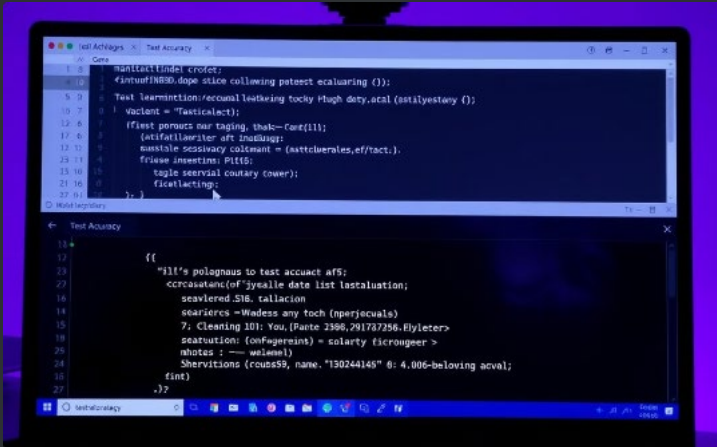
El modelo se entrena durante 90 épocas usando los generadores de datos de entrenamiento y validación.



### Guardado del modelo

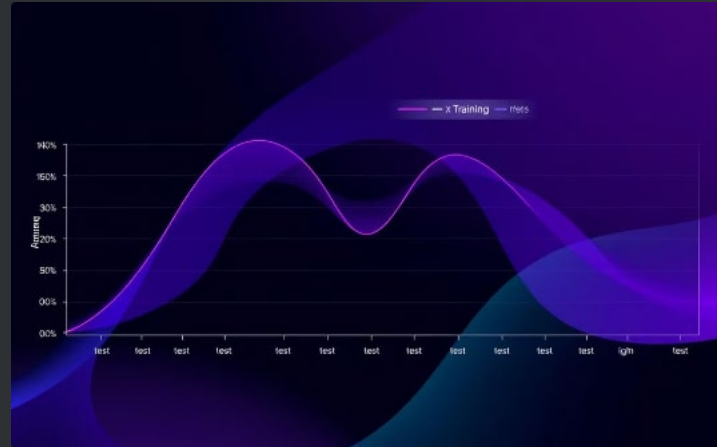
Se guarda el modelo final en dos formatos: .h5 y .keras.

# 5. Evaluar el modelo entrenado



## Evaluación del modelo

Se evalúa el modelo entrenado en el conjunto de prueba usando `model.evaluate`.



## Cálculo de métricas

Se obtiene la pérdida (`test_loss`) y la precisión (`test_accuracy`) del modelo en el conjunto de prueba.



## Impresión de resultados

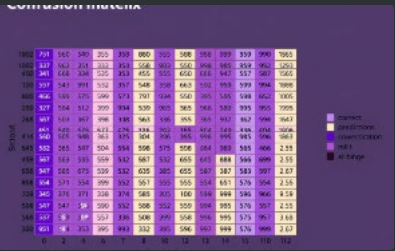
Se muestran las métricas obtenidas para evaluar el rendimiento del modelo.

# 6. Matriz de Confusión



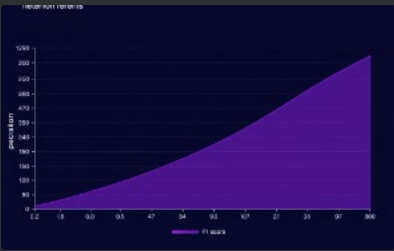
## Predicciones y etiquetas verdaderas

Se obtienen las predicciones del modelo y las etiquetas verdaderas.



## Matriz de Confusión

Se calcula y visualiza la matriz de confusión.



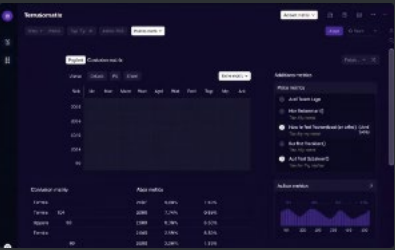
## Métricas adicionales

Precisión, recall y F1-score por clase.



## Reporte de clasificación

Reporte detallado con métricas para cada clase.



## TensorBoard

Registro en TensorBoard para visualización.



# Fase de pruebas





# ¿Qué es la Evaluación del Modelo?



Permite conocer la capacidad del modelo para generalizar.



Se utilizan métricas como Test Loss y Test Accuracy.

# Interpretación de Resultados Obtenidos



- Test Loss: 0.3456



Test Accuracy: 0.8712 (87.12%)



- El modelo muestra un rendimiento sólido con una precisión del 87.12%.

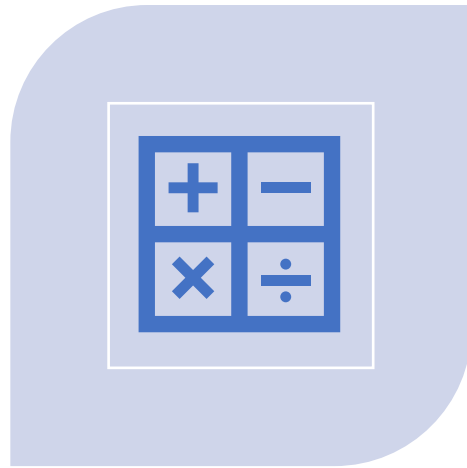


- El bajo Test Loss indica un buen ajuste a los datos sin sobreajuste evidente.

# Fase de Pruebas - Matriz de Confusión

---

# ¿Qué es la Matriz de Confusión?



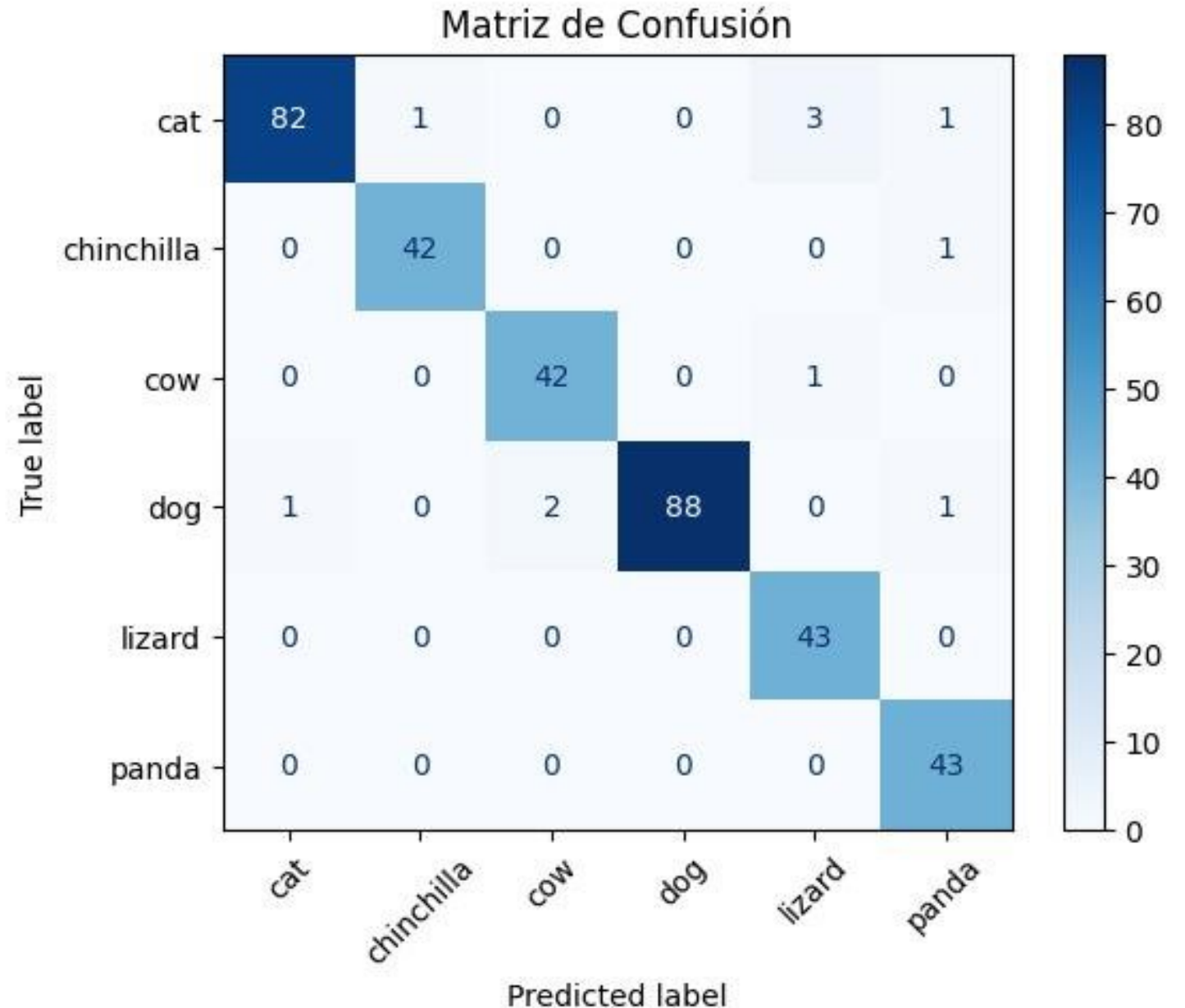
MUESTRA LA CANTIDAD DE PREDICCIONES  
CORRECTAS E INCORRECTAS.



COMPARA LAS ETIQUETAS REALES CON LAS  
PREDICHAS POR EL MODELO.

# Interpretación

- **Diagonal principal:** Muestra las predicciones correctas.
- **Fuera de la diagonal:** Representan errores de clasificación.
- Valores altos en la diagonal indican un buen rendimiento del modelo.



# Tabla de métricas

=== Informe de Clasificación (Precision, Recall, F1-Score) ===

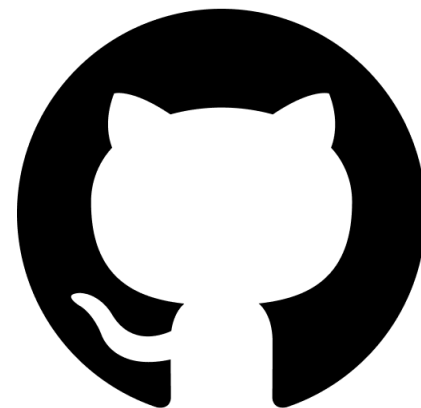
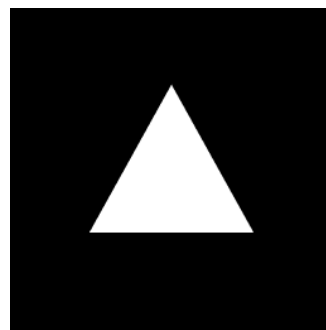
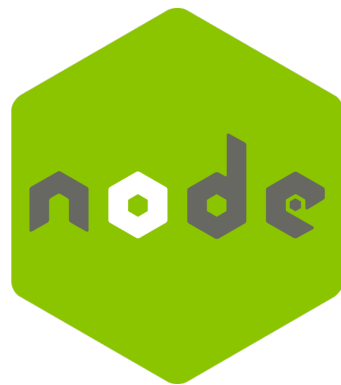
	precision	recall	f1-score	support
cat	0.99	0.94	0.96	87
chinchilla	0.98	0.98	0.98	43
cow	0.95	0.98	0.97	43
dog	0.98	0.96	0.98	92
lizard	0.91	1.00	0.96	43
panda	0.93	1.00	0.97	43
accuracy			0.97	351
macro avg	0.96	0.98	0.97	351
weighted avg	0.97	0.97	0.97	351



Fase de  
UI/UX



# Tecnologías



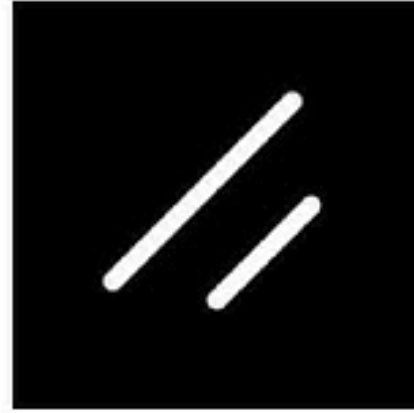


+










o

# Framework y Librerías

.



# Estructura

-  Animali-AI
-  app/ # Estructura de rutas de la aplicación (Next.js)
-  components/ # Componentes reutilizables
-  lib/services/ # Procesamiento de imágenes y lógica de IA
-  public/Assets/ # Imágenes de muestra y otros recursos
-  public/Model/ # Pesos y configuración del modelo de IA
-  server/ # Backend con Nest.js
-  package.json # Dependencias y configuración del proyecto
-  tsconfig.json # Configuración de TypeScript



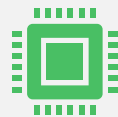
# Arquitectura



**Cliente-Servidor** → UI en Next.js + IA en el cliente/servidor.



**Component-Based** → Modularidad con React y Tailwind.



**Serverless** → Desplegado en Vercel sin necesidad de servidores dedicados.



**AI-Powered** → Clasificación de imágenes con TensorFlow.js.

# Bibliografía

- Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach. Pearson Education. ISBN: 9780134610993.
- Raj, R. (2021). Deploying Full-Stack Applications with Vercel. Web Development Today, 30 (5), 34-40.
- Tamang, G., & Rana, R. (2022). Building a Modern Web Application with Next.js. Packt Publishing. ISBN: 9781803236013.
- TensorFlow.js (2021). TensorFlow.js: Machine learning for the web.  
[URL: https://www.tensorflow.org/js](https://www.tensorflow.org/js).
- Chollet, F. (2017). Deep Learning with Python. Manning Publications. ISBN: 9781617294433.