VISIÓN POR COMPUTADORA 3

Informe del trabajo práctico

Dic 2025

DEGANO, MYRNA LORENA N° SIU a1618 myrna.l.degano@gmail.com
GUSTAVO JULIÁN RIVAS N° SIU a1620 gus.j.rivas@gmail.com

Índice

Introducción

El presente informe describe en detalle el proyecto desarrollado con el objetivo de entrenar y evaluar un modelo de **visión por computadora** para la tarea de **clasificación de imágenes**, específicamente diferenciando entre gatos y perros.

El proyecto incluye la adquisición y preparación de datos, el **fine-tuning** de un modelo preentrenado, la evaluación del desempeño y la construcción de una aplicación para inferencia.

La clasificación de imágenes constituye una de las tareas fundamentales del aprendizaje profundo. En particular, en este proyecto se utiliza el conjunto de datos "**Cats vs Dogs**" de **Hugging Face** que se ha convertido en un estándar para evaluar modelos de visión debido a su dificultad moderada, su representatividad y su disponibilidad pública.

Al abordar este problema mediante el modelo **MobileViT** se busca explotar un equilibrio interesante entre eficiencia computacional y desempeño en tareas visuales.

Este informe detalla la arquitectura general del sistema, las herramientas utilizadas, las decisiones de diseño, la implementación técnica, las métricas de evaluación y los resultados obtenidos. Asimismo, se presenta una sección de conclusiones finales y propuestas de mejora futura.

Objetivos

El objetivo principal del proyecto es desarrollar un sistema completo basado en aprendizaje profundo capaz de:

1. **Construir un pipeline reproducible** para la descarga, limpieza, preprocesamiento y preparación de un dataset de imágenes.
2. **Aplicar fine-tuning a un modelo MobileViT** para adaptarlo a la tarea de clasificación de gatos versus perros.
3. **Entrenar y evaluar el modelo** utilizando métricas estándar.
4. **Proveer una aplicación simple de inferencia**, donde el usuario puede cargar una imagen y recibir una predicción del modelo entrenado como prototipo o demo preliminar.
5. **Documentar los resultados obtenidos** y establecer bases para futuras mejoras o extensiones.

El proyecto tiene un carácter formativo, técnico y experimental, enfocado en entender la dinámica del entrenamiento de modelos modernos de visión y en demostrar una implementación operativa de punta a punta.

Arquitectura general del sistema

La arquitectura del sistema está organizada de manera **modular**, permitiendo separar responsabilidades y facilitar el mantenimiento.

Cada módulo debe cumplir **una única tarea o responsabilidad claramente definida**. Aplicar este principio permite:

- **Separar el código según su propósito**, evitando que un único script acumule lógica de descarga de datos, entrenamiento y evaluación simultáneamente.
- **Reducir la complejidad**, ya que al aislar responsabilidades se evita que una modificación en un proceso afecte accidentalmente otros.
- **Incrementar la legibilidad**, permitiendo entender el rol de cada parte del sistema sin necesidad de examinar un archivo extenso o monolítico.
- **Facilitar el mantenimiento,** ya que permite depurar cada componente de manera individual, realizar pruebas unitarias específicas para funciones o procesos concretos y actualizar partes del sistema sin afectar otras.
- **Generar componentes reutilizables** en futuros proyectos o experimentos, por ejemplo, con diferentes conjuntos de datos.
- **Organizar el trabajo del equipo**, ya que diferentes personas pueden trabajar en partes distintas del proyecto sin interferencias, facilitando el trabajo colaborativo.

Estructura de archivos

configs/	Esta carpeta cumple un rol fundamental en la organización del proyecto, ya que centraliza todos los parámetros, valores configurables y opciones de ejecución que determinan el comportamiento del sistema.
config.py	Se encarga de gestionar la carga centralizada de los parámetros de configuración del proyecto. Toma el archivo *config.yaml*, ubicado en la misma carpeta, e interpreta su contenido para convertirlo en un diccionario Python accesible desde cualquier parte del sistema. De este modo, todos los scripts pueden acceder a una única fuente de configuración consistente, lo que favorece la modularidad, la reproducibilidad y el mantenimiento del proyecto.
config.yaml	Centraliza todos los parámetros operativos del proyecto, permitiendo definir de manera clara, editable y reproducible cada aspecto del entrenamiento, evaluación e inferencia.

Principales parámetros:

- Relacionados con el conjunto de datos (Sección **`dataset`**):
  - ✓ **`name`**
    Nombre del dataset que será utilizado, en este caso `"cats_vs_dogs"`, correspondiente a un conjunto de imágenes alojado en Hugging Face. Este parámetro indica qué conjunto de datos debe descargarse y procesarse para entrenar y evaluar el modelo.
  - ✓ **`batch_size`**
    Tamaño del lote (batch) que se utiliza en cada iteración del entrenamiento. En este caso, 16, que significa que el modelo procesa 16 imágenes por paso. Esto afecta directamente el uso de memoria, la estabilidad del entrenamiento y la velocidad.
  - ✓ **`val_split`**
    Proporción del dataset que se reservará para validación, en este caso 0.2 (20%). Esta partición es fundamental para evaluar el rendimiento del modelo sin mezclar datos que fueron utilizados para entrenarlo.
  - ✓ **`seed`**
    Semilla aleatoria usada para asegurar reproducibilidad. Controla procesos aleatorios como la división entre entrenamiento/validación, garantizando que los experimentos puedan repetirse con los mismos resultados.
  - ✓ **`max_samples`**
    Número máximo de muestras a utilizar del dataset. Se emplea para reducir el tamaño de los datos en casos donde se desea acelerar el entrenamiento o realizar pruebas rápidas sin procesar el dataset completo.

    Nota sobre el tamaño del dataset: Debido a la limitada disponibilidad de recursos de cómputo, no se utiliza el dataset completo. Se realizaron experimentos con diferentes

números de `max_samples` y se determinó que 5000 imágenes son suficientes para obtener un balance entre entrenamiento rápido y buen desempeño de inferencia.

- Relacionados con el entrenamiento (Sección **`training`**):
  - ✓ **`epochs`**
    Cantidad total de pasadas completas del dataset de entrenamiento a través del modelo. Se estableció en 20, pero finalmente el modelo completa el entrenamiento en menos épocas por el uso de **early stopping**, evitando seguir entrenando innecesariamente cuando el modelo ya no está aprendiendo.
  - ✓ **`learning_rate`**
    Tasa de aprendizaje del optimizador. Define el tamaño del paso que da el modelo al actualizar los pesos. Un valor muy alto puede volver inestable el entrenamiento; uno muy bajo puede hacerlo lento. Utilizamos 0.00005 como valor inicial.
  - ✓ **`weight_decay`**
    Parámetro de regularización que penaliza valores muy grandes en los pesos del modelo. Ayuda a prevenir el sobreajuste y mejora la capacidad de generalización. Utilizamos 0.1.
  - ✓ **`patience`**
    Número de épocas que deben pasar sin mejora antes de activar el *early stopping*. Lo definimos en 5 y esto determinó que el modelo termine de entrenar antes de las 20 épocas preestablecidas.
  - ✓ **`logging_steps`**
    Indica cada cuántas épocas se registrarán métricas o mensajes de progreso. Lo establecimos en 1, para obtener información en cada época del entrenamiento.

- Relacionados con el seguimiento (Sección **`mlflow`**):

  Nota sobre la herramienta de tracking: Utilizamos MLflow para el registro de los diferentes experimentos y sus respectivas métricas y artefactos. Esta sección permite configurar sus parámetros.

  - ✓ **`tracking_uri`**
    Ruta que utilizará MLflow para el almacenamiento de los experimentos, métricas y modelos. En este caso, utilizamos un URI local (file:./mlflow/mlruns) porque no disponemos de un servidor MLflow remoto. Esta carpeta luego la montamos en un servidor MLflow para poder observar los resultados.
  - ✓ **`experiment_name`**
    Nombre del experimento dentro de MLflow. Permite agrupar corridas bajo una misma categoría, facilitando la comparación entre diferentes entrenamientos.

- Relacionados con la inferencia (Sección **`inference`**):

  ✓ **`class_names`**

  Lista que define los nombres interpretables de las clases del modelo. En este caso ["Cat", "Dog"]. Es usada por la aplicación de inferencia para mostrar resultados comprensibles para el usuario final.
  Permite que esto se personalice en caso de realizar experimentos con otros conjuntos de datos.

- Relacionados con el modelo (Sección **`model`**):

  ✓ **`name`**

  Modelo base a utilizar. Este parámetro indica qué arquitectura se cargará desde Hugging Face. En este caso: ***apple/mobilevit-small***.

  ✓ **`num_classes`**

  Cantidad de clases que debe predecir el modelo final. Para nuestro dataset el valor es 2. En caso de usar otro dataset, se puede modificar y este valor determina la dimensión de la última capa.

  Nota sobre la elección del modelo: La elección se fundamenta en una combinación de factores técnicos, de rendimiento y de eficiencia que lo convierten en una arquitectura especialmente adecuada para este problema. MobileViT es un modelo híbrido que integra operaciones convolucionales ligeras —propias de arquitecturas móviles como MobileNet— con bloques tipo Transformer que permiten capturar dependencias globales en la imagen. Esto le otorga una ventaja significativa frente a modelos puramente convolucionales, ya que combina **bajo costo computacional** con **alta capacidad de representación**.

  La variante **"small"** presenta además beneficios importantes:

  1. **Eficiencia computacional:** MobileViT-Small está diseñado para ejecutarse en dispositivos con recursos limitados, (como es nuestro caso) por lo que su tamaño reducido y su bajo consumo de memoria permiten entrenarlo y realizar inferencias sin necesidad de hardware de gran escala.

  2. **Rendimiento competitivo:** A pesar de su tamaño compacto, MobileViT ofrece un rendimiento comparable al de modelos más pesados en tareas de visión. Su integración de mecanismos Transformer mejora la capacidad del modelo para capturar relaciones espaciales globales, lo cual resulta útil en imágenes donde los animales pueden aparecer en distintas posiciones o contextos visuales.

3. **Adecuación al dataset elegido:** El problema es relativamente sencillo en comparación con datasets complejos; por lo tanto, un modelo pequeño pero expresivo es suficiente para lograr una excelente precisión sin incurrir en costos computacionales innecesarios. MobileViT-Small ofrece el equilibrio perfecto entre capacidad y eficiencia para este tipo de problemas de clasificación.

4. **Rapidez en el fine-tuning:** La arquitectura ligera permite realizar entrenamientos más rápidos, reduciendo tiempos de iteración y permitiendo ajustar hiperparámetros de manera más ágil. Esto favorece el enfoque experimental del proyecto y acelera la obtención de resultados.

5. **Escalabilidad y extensibilidad:** Al tratarse de una familia de modelos con variantes de distinto tamaño (XS, S, M), permite escalar a versiones mayores si en el futuro se desea abordar datasets más complejos sin abandonar la misma arquitectura conceptual.

- Relacionados con el preprocesamiento de las imágenes (Sección `transforms`):

  ✓ `image_size`
  Tamaño al que se redimensionarán las imágenes antes de ser procesadas por el modelo (224×224 píxeles, tamaño esperado por MobileViT)
  ✓ `rotation`
  Máximo grado de rotación empleado en augmentation (15). Introduce variabilidad en los datos, ayudando al modelo a generalizar mejor frente a diferentes orientaciones de las imágenes.
  ✓ `escalation`
  Factor de escala aplicado durante las transformaciones. Permite simular zoom in/out sobre la imagen, agregando variaciones beneficiosas para la robustez del modelo.

- Relacionados con la organización de los archivos del proyecto (Sección `folders`):

  ✓ `mlruns`
  Carpeta donde MLflow almacenará los experimentos y registros generados durante el entrenamiento.
  ✓ `test_dataset`
  Carpeta donde se guardan los datos destinados a validación del modelo entrenado.
  ✓ `root`
  Ruta raíz del proyecto. Facilita crear rutas relativas a partir de un único punto de referencia.

- ✓ **`model`**

  Carpeta donde se guardará el mejor modelo encontrado durante el entrenamiento para ser usado luego en pruebas de inferencia.

🗀 notebooks/	
EDA.ipynb	Notebook donde se realizó una breve exploración inicial del dataset para entender qué tipo de imágenes contiene.
Train.ipynb	Dada la limitada disponibilidad de servidores y recursos suficientes de cómputo, se utilizaron notebooks en Google Colab para pruebas preliminares y validaciones del código de entrenamiento y evaluación. De todas formas, el código está modularizado por completo en archivos *.py, por lo que, no requiere de notebooks para su ejecución.
Eval.ipynb	
Metrics.ipynb	Notebook que toma los archivos de métricas logueados en MLflow y genera dashboard interactivo custom.

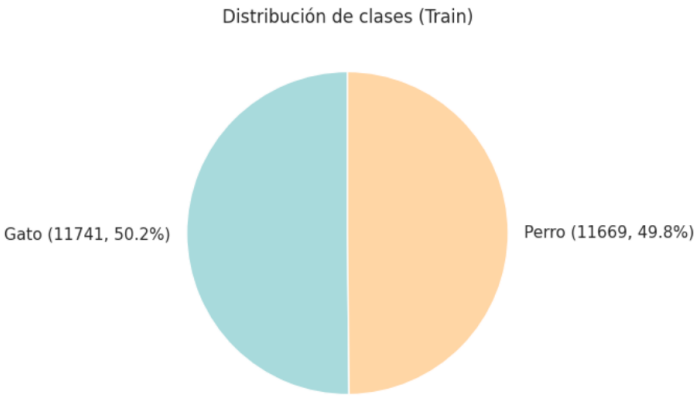
Notas sobre el análisis de los datos del dataset:

- Contiene imágenes de **gatos y perros** para clasificación binaria.
- Tiene un total de **23.410 ejemplos**. Cada ejemplo consta de dos campos principales:
  - image: la imagen (decodificada como un objeto PIL)
  - labels: etiqueta entera indicando clase — 0 para “cat”, 1 para “dog”.

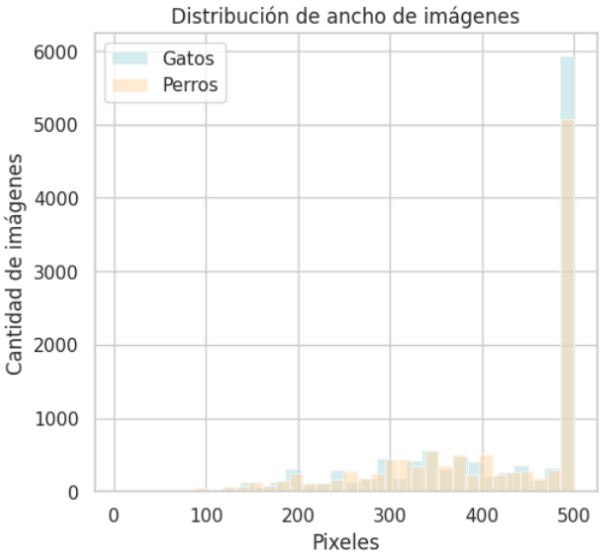
Las imágenes provienen originalmente de un dataset mayor, que fue recolectado mediante la colaboración de un sitio de adopción de mascotas, lo que asegura que las imágenes provienen de contextos reales de gatos y perros bajo distintos fondos, poses, iluminación, etc.
Este dataset ya tiene un historial de uso en la comunidad de ML / visión por computadora como base para tareas de clasificación.

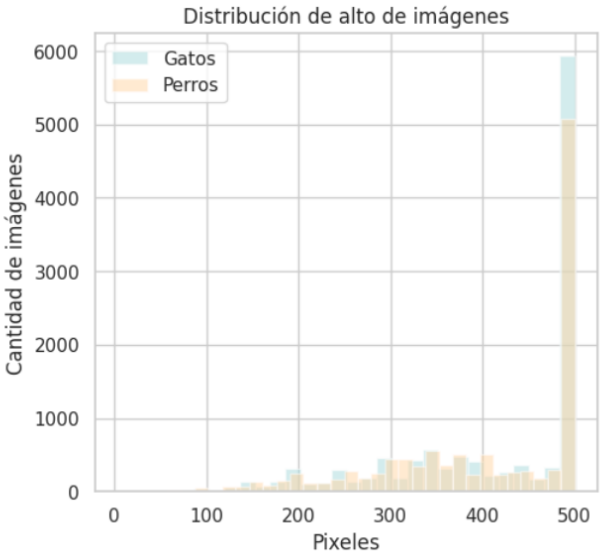
Se observa que las clases están bien balanceadas (todas bajo la clave ‘train’).
Además se observan que tienen características casi óptimas para el entrenamiento, por lo que, no necesitan demasiado preprocesamiento.

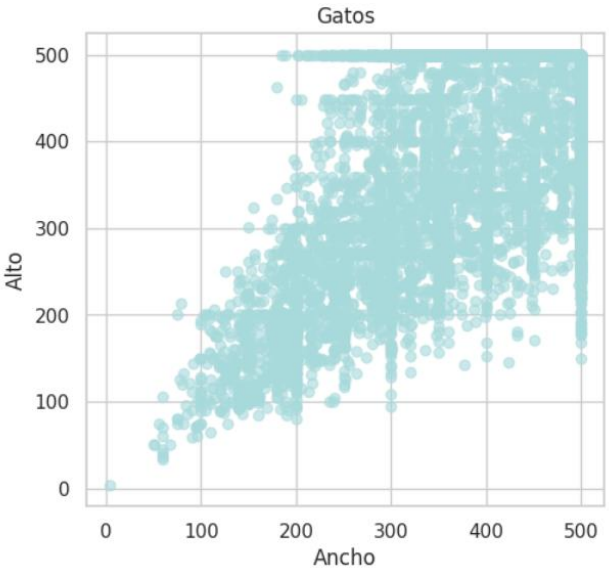
Se exploraron igualmente algunas transformaciones útiles para problemas de clasificación, para complementar, ya que el modelo elegido, realiza internamente *reescalado, redimensionado y reordenado de canales (RGB→BGR).*


Distribución de clases (Train)
Gato (11741, 50.2%)
Perro (11669, 49.8%)


Histograma de brillo promedio
2500
2000
1500
1000
500
0
50
100
150
200
250


Distribución de ancho de imágenes
Gatos
Perros
Cantidad de imágenes
6000
5000
4000
3000
2000
1000
0
0
100
200
300
400
500
Pixeles


Distribución de alto de imágenes
Gatos
Perros
Cantidad de imágenes
6000
5000
4000
3000
2000
1000
0
0
100
200
300
400
500
Pixeles


Gatos
Alto
500
400
300
200
100
0
0
100
200
300
400
500
Ancho

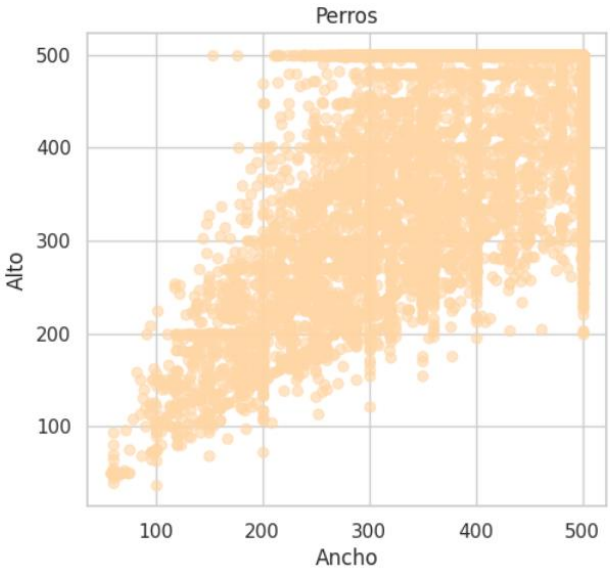
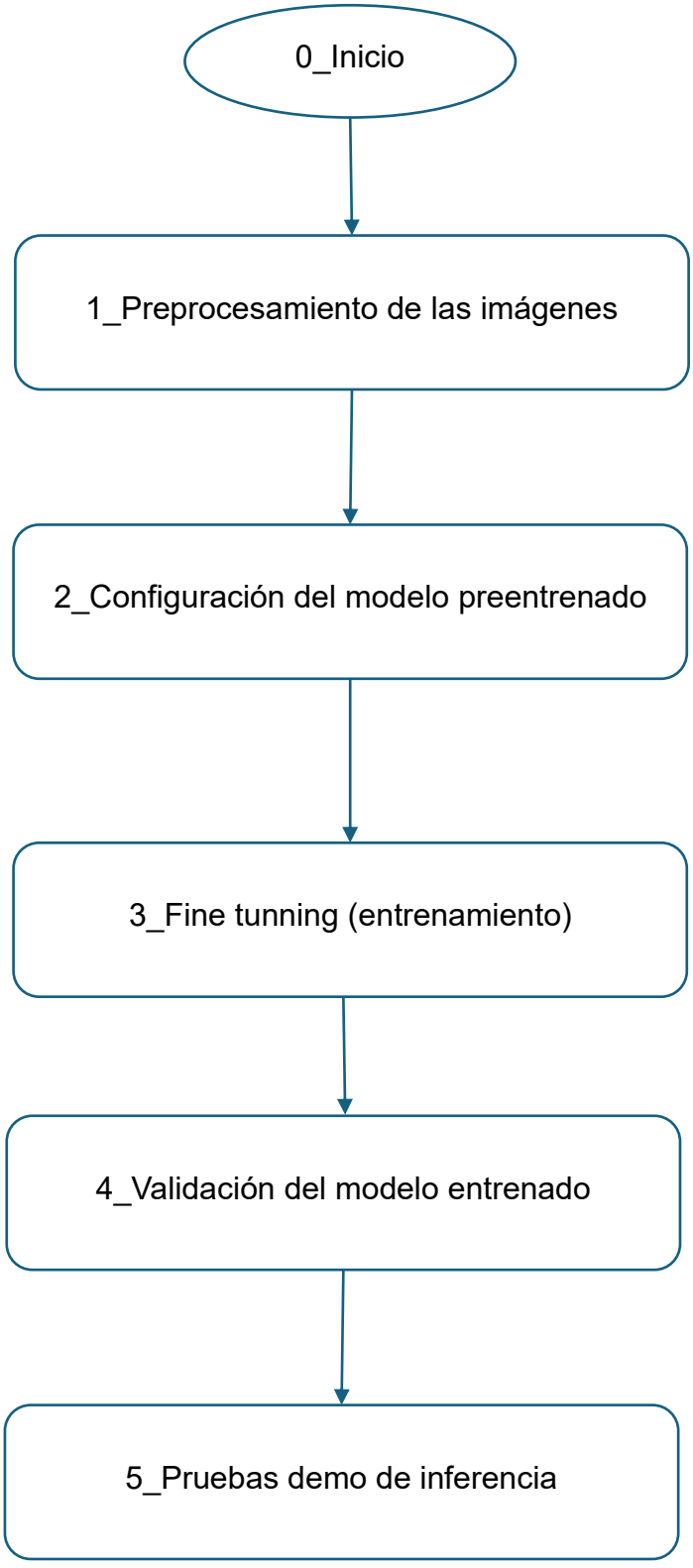

Perros
Alto
500
400
300
200
100
100
200
300
400
500
Ancho

Diagrama de flujo



- Los requerimientos para el entorno de ejecución están en **requirements.txt**

- Además, como se explicó previamente, todos los parámetros se cargan como módulo desde **configs/** de forma tal que pueden ser ajustados de forma centralizada según la necesidad.

- El módulo de entrenamiento está en **training/** cuyo archivo principal es **train.py**

  - **train.py**
    Script principal que coordina todo el flujo de entrenamiento*:

    1. Cargar del dataset
       data_loader.py
       - ✓ Carga el dataset de Hugging Face.
       - ✓ Divide los datos en conjuntos de entrenamiento y validación.

    2. Aplicar transformaciones necesarias
       augmentations.py
       - ✓ Define las transformaciones de data augmentation (rotaciones, flips, recortes, redimensionamiento). Esto ayuda a que el modelo generalice mejor y evita el sobreajuste.

    3. Preprocesamiento de las imágenes
       preprocessing.py
       - ✓ Aplica las transformaciones y convierte a tensores dejando los datos listos para el procesamiento en el formato compatible con el modelo.

    4. Configuración del modelo
       model_builder.py
       - ✓ Configura MobileViT preentrenado y reemplaza la cabeza final de acuerdo a la tarea que debe realizar.
       - ✓ Permite ajustar hiperparámetros del modelo.

    5. Parámetros de entrenamiento
       trainer_utils.py
       - ✓ Define funciones auxiliares para el entrenamiento.

       callbacks.py
       - ✓ Callbacks que gestionan guardado de checkpoints y logging de métricas.

    6. Entrenamiento
       (main)*

7. Registro en MLflow

   mlflow_utils.py

   ✓ Maneja la integración con MLflow para registrar experimentos y artefactos.

   ✓ Permite reproducir entrenamientos y comparar resultados de forma organizada.

Ejemplo de entrenamiento para 5000 imágenes y 20 épocas:

```
README.md: 8.16k/? [00:00<00:00, 747kB/s]
data/train-00000-of-00002.parquet: 100% 330M/330M [00:02<00:00, 198MB/s]
data/train-00001-of-00002.parquet: 100% 391M/391M [00:04<00:00, 88.9MB/s]
Generating train split: 100% 23410/23410 [00:05<00:00, 7414.22 examples/s]
Saving the dataset (1/1 shards): 100% 1000/1000 [00:00<00:00, 7527.37 examples/s]
preprocessor_config.json: 100% 187/187 [00:00<00:00, 18.7kB/s]
config.json: 70.0k/? [00:00<00:00, 5.89MB/s]
pytorch_model.bin: 100% 22.5M/22.5M [00:01<00:00, 22.4MB/s]
```

```
Map: 100% 4000/4000 [02:17<00:00, 28.05 examples/s]
model.safetensors: 100% 22.4M/22.4M [00:00<00:00, 35.8MB/s]
Map: 100% 1000/1000 [00:38<00:00, 25.65 examples/s]
```

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.157700	0.093633	0.988000	0.987952	0.991935	0.984000
2	0.236500	0.046884	0.989000	0.989033	0.986083	0.992000
3	0.232000	0.033288	0.989000	0.988967	0.991952	0.986000
4	0.051000	0.040848	0.986000	0.985887	0.993902	0.978000
5	0.013700	0.033016	0.991000	0.990955	0.995960	0.986000
6	0.002200	0.029256	0.991000	0.990991	0.991984	0.990000
7	0.001300	0.032911	0.991000	0.990991	0.991984	0.990000
8	0.005900	0.031588	0.989000	0.989011	0.988024	0.990000
9	0.006700	0.034485	0.990000	0.989960	0.993952	0.986000
10	0.011200	0.025860	0.991000	0.990991	0.991984	0.990000
11	0.012700	0.034328	0.991000	0.990991	0.991984	0.990000
12	0.000500	0.041501	0.988000	0.988024	0.986056	0.990000
13	0.000500	0.032785	0.990000	0.990000	0.990000	0.990000
14	0.000800	0.054305	0.989000	0.988922	0.995943	0.982000
15	0.001800	0.064856	0.989000	0.988922	0.995943	0.982000

Con patience=5, el entrenamiento finaliza a las 15 épocas ya que se aplica el callback de **Early Stopping** porque, de acuerdo al valor de la función de pérdida, el aprendizaje deja de mejorar.


```
Epoch 1 - Metrics so far: {'loss': 0.1577, 'grad_norm': 2.3971755504608154, 'learning_rate': 4.7510000000000004e-05, 'epoch': 1.0, 'step'
Epoch 2 - Metrics so far: {'loss': 0.2365, 'grad_norm': 8.661844253540039, 'learning_rate': 4.5010000000000004e-05, 'epoch': 2.0, 'step':
Epoch 3 - Metrics so far: {'loss': 0.232, 'grad_norm': 29.250919342041016, 'learning_rate': 4.251e-05, 'epoch': 3.0, 'step': 750}
Epoch 4 - Metrics so far: {'loss': 0.051, 'grad_norm': 5.20906925201416, 'learning_rate': 4.0010000000000005e-05, 'epoch': 4.0, 'step': 1(
Epoch 5 - Metrics so far: {'loss': 0.0137, 'grad_norm': 5.025270462036133, 'learning_rate': 3.751e-05, 'epoch': 5.0, 'step': 1250}
Epoch 6 - Metrics so far: {'loss': 0.0022, 'grad_norm': 0.046936482191085815, 'learning_rate': 3.5010000000000005e-05, 'epoch': 6.0, 'ste
Epoch 7 - Metrics so far: {'loss': 0.0013, 'grad_norm': 0.021507926285266876, 'learning_rate': 3.251e-05, 'epoch': 7.0, 'step': 1750}
Epoch 8 - Metrics so far: {'loss': 0.0059, 'grad_norm': 1.755413293838501, 'learning_rate': 3.001e-05, 'epoch': 8.0, 'step': 2000}
Epoch 9 - Metrics so far: {'loss': 0.0067, 'grad_norm': 5.165678977966309, 'learning_rate': 2.7510000000000003e-05, 'epoch': 9.0, 'step':
Epoch 10 - Metrics so far: {'loss': 0.0112, 'grad_norm': 1.1104822158813477, 'learning_rate': 2.501e-05, 'epoch': 10.0, 'step': 2500}
Epoch 11 - Metrics so far: {'loss': 0.0127, 'grad_norm': 15.570735931396484, 'learning_rate': 2.251e-05, 'epoch': 11.0, 'step': 2750}
Epoch 12 - Metrics so far: {'loss': 0.0005, 'grad_norm': 0.005604459438472986, 'learning_rate': 2.001e-05, 'epoch': 12.0, 'step': 3000}
Epoch 13 - Metrics so far: {'loss': 0.0005, 'grad_norm': 0.005690644029527903, 'learning_rate': 1.751e-05, 'epoch': 13.0, 'step': 3250}
Epoch 14 - Metrics so far: {'loss': 0.0008, 'grad_norm': 0.00877635832875967, 'learning_rate': 1.5010000000000002e-05, 'epoch': 14.0, 'st
Epoch 15 - Metrics so far: {'loss': 0.0018, 'grad_norm': 0.014861617237329483, 'learning_rate': 1.2509999999999999e-05, 'epoch': 15.0, 'st
```

Se observa una clara mejora en las primeras épocas (descenso de la curva en la función de pérdida), pero luego ya la curva se estabiliza.

- El módulo de evaluación está en **evaluation/** cuyo archivo principal es **evaluate.py**

  - **evaluate.py**
    Script principal que coordina el proceso de evaluación.
    Utiliza además:
    a) **metrics.py**
       Define las métricas a calcular para el problema de clasificación.
    b) **plots.py**
       Genera gráficas auxiliares que permiten evaluar el desempeño y ser guardadas en MLflow como artefactos complementarios al modelo.

Los resultados de las métricas sobre el dataset de validación resultaron ser muy buenos:

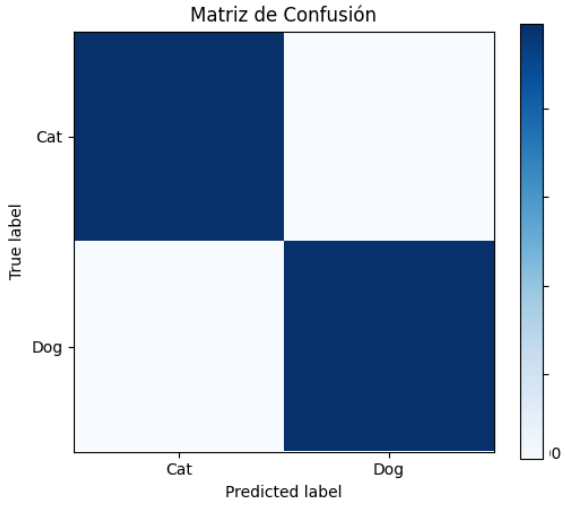
```
Dataset loading...


Metrics evaluation...

Eval Dataset Metrics:
eval_accuracy: 0.99
eval_f1: 0.99
eval_precision: 0.99
eval_recall: 0.99
```

En este caso, todas las métricas son muy altas (0.99), lo que indica que el modelo distingue muy bien entre gatos y perros, con muy pocos errores de clasificación.

Métrica	**Qué mide / para qué sirve**	**Comentarios**
Accuracy	Proporción de predicciones correctas sobre el total de casos.	El 99% de todas las imágenes evaluadas (gatos y perros) fueron clasificadas correctamente.
Precision	Qué proporción de predicciones positivas son realmente correctas.	La clase positiva es “perro” (1), por lo que, de todas las veces que el modelo dijo “perro”, el 99% eran realmente perros.
Recall	Qué proporción de positivos reales fueron detectados.	De todos los perros reales, el modelo identificó correctamente al 99%.
F1 Score	Promedio armónico entre precisión y recall; balancea ambos.	El modelo tiene un excelente balance (99%) entre identificar perros correctamente y no equivocarse con los gatos en la predicción.

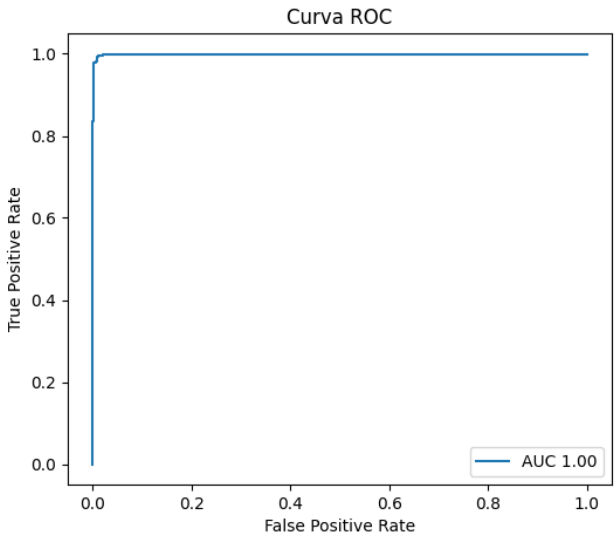
Adicionalmente a estas métricas se graficó:

- **Matriz de confusión**
  Tabla que muestra TP (verdaderos positivos), TN (verdaderos negativos), FP (falsos positivos), FN (falsos negativos) y permite ver errores específicos por clase.



Casi todos los gatos fueron clasificados como gatos y casi todos los perros como perros, apenas **1% de errores**.

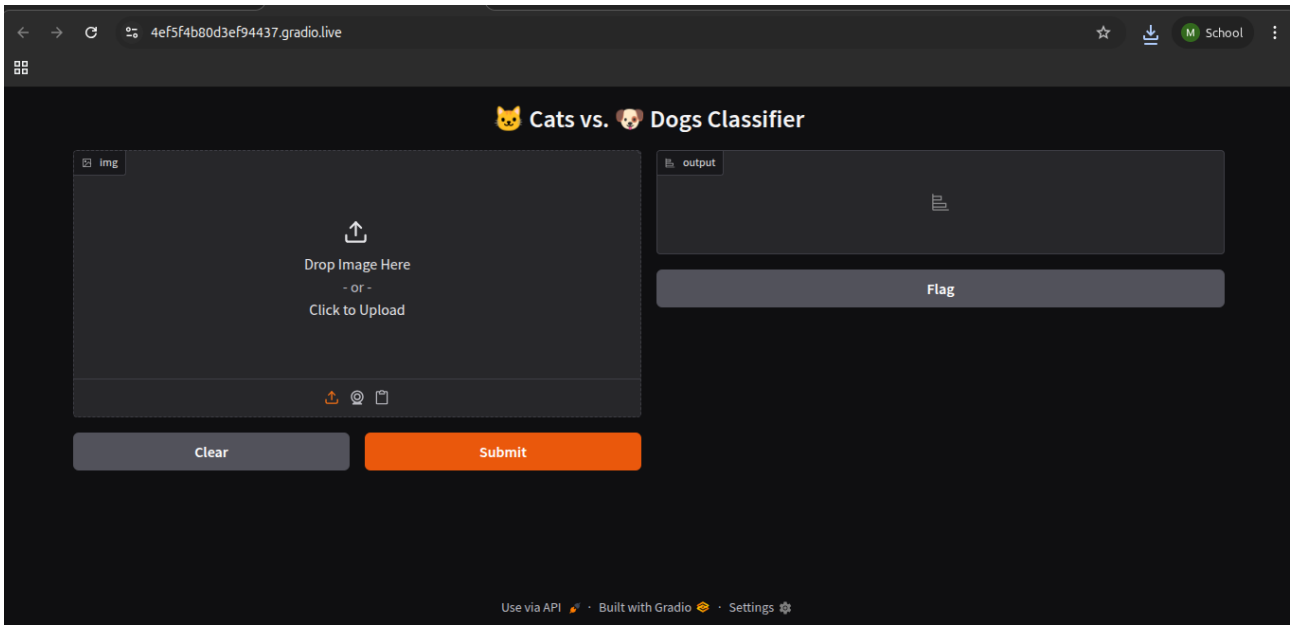
- **Curva ROC**
  Evalúa el rendimiento del modelo para distintos umbrales de decisión; AUC cercano a 1 indica excelente separación entre clases.



El modelo separa casi perfectamente gatos de perros (área bajo la curva casi 1).

- El módulo de inferencia está en **inference/**

  - **app.py**
    - ✓ El objetivo principal de este código es crear una **interfaz web interactiva** que permita a cualquier usuario subir imágenes y obtener la predicción de si corresponde a un gato o un perro, sin necesidad de usar directamente código o terminal.
    - ✓ Esta interfaz está lejos de ser una aplicación real productiva, simplemente facilita la demostración y prueba del modelo entrenado de manera accesible y visual.
    - ✓ Para implementarla se utiliza **Gradio**, una biblioteca de Python diseñada para construir rápidamente aplicaciones web interactivas para modelos de machine learning. Permite definir fácilmente los tipos de entrada y salida, manejar la ejecución de la función de predicción y mostrar los resultados en **tiempo real**.



MLFlow

Dado que no tenemos acceso a un servidor MLflow remoto, decidimos usar para este proyecto el enfoque de almacenar en file system local los experimentos de MLflow durante la ejecución, y luego crear un entorno que nos permita visualizar los resultados con MLflow UI.

En la carpeta **mlflow/** se encuentran los archivos de configuración para la creación de un entorno docker con MLflow.

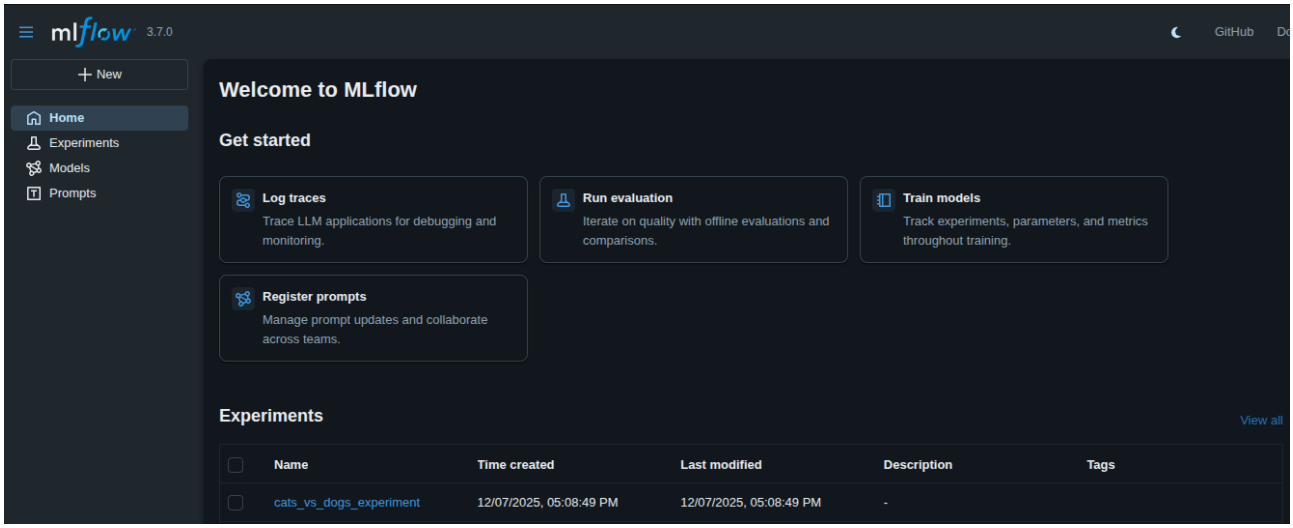
- .env
- Dockerfile
- docker-compose.yml
- run_mlflow.sh

La carpeta **mlruns**/ contiene los outputs almacenados durante entrenamiento y evaluación.

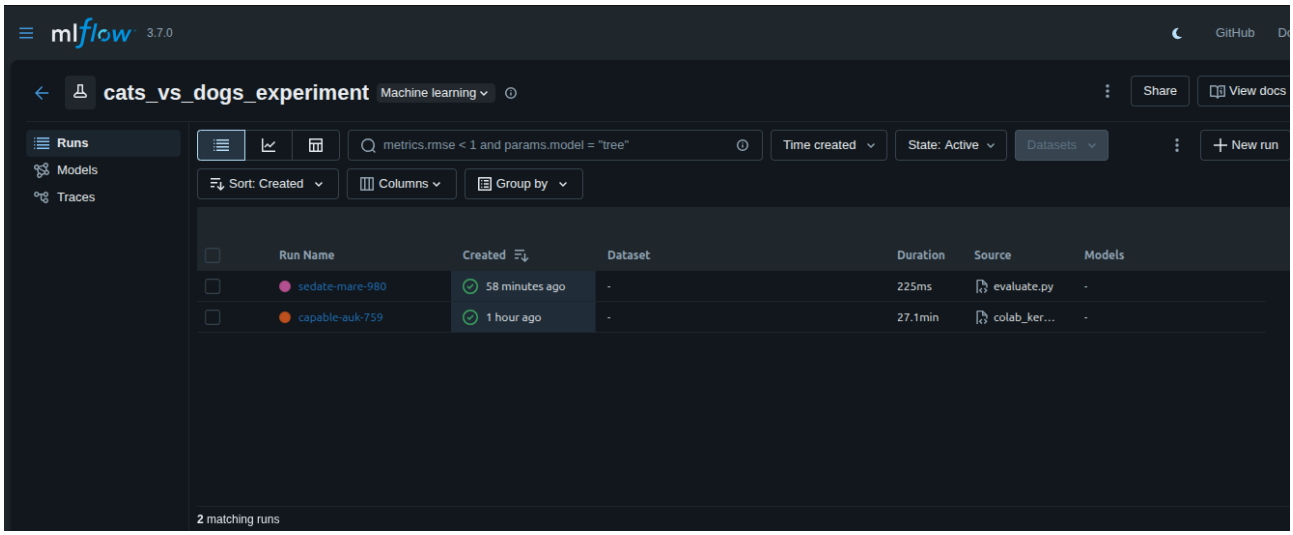
Ejecutando **run_mlflow.sh** creamos nuestro propio entorno de MLflow montando la carpeta mlruns para visualizar los resultados.

```
>> Levantando contenedor MLflow...
[+] Building 0.0s (0/0)                                                    docker:default
[+] Running 2/2
 ✔ Network mlflow_default   Created                                                 0.5s
 ✔ Container mlflow_server  Started                                                 2.7s
>> Esperando al healthcheck...
   Esperando... (1/20)
   Esperando... (2/20)
   Esperando... (3/20)
   Esperando... (4/20)
   Esperando... (5/20)
>> MLflow está healthy ✔
================================================
 MLflow UI disponible en:
  http://localhost:5000
```

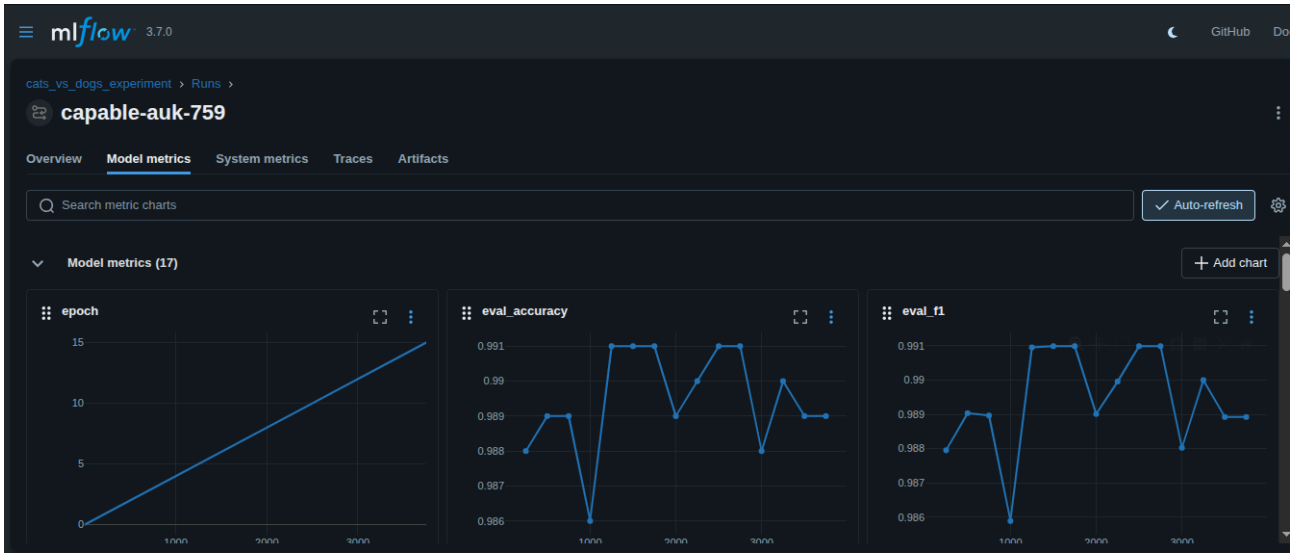
Se visualiza el experimento “cats_vs_dogs_experiment” creado para el proyecto:

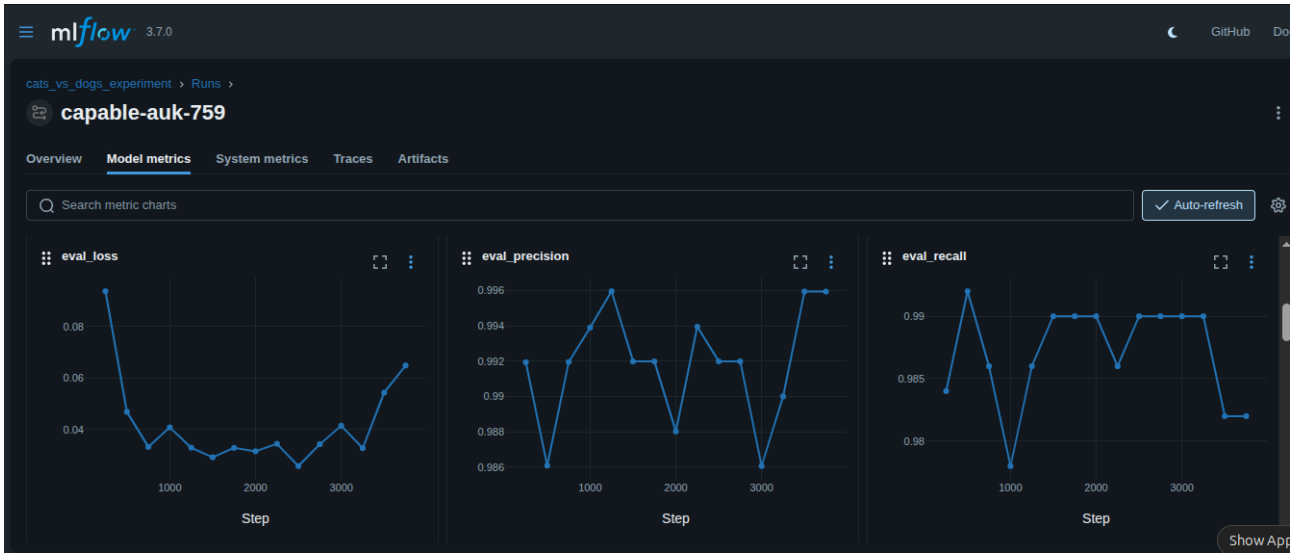


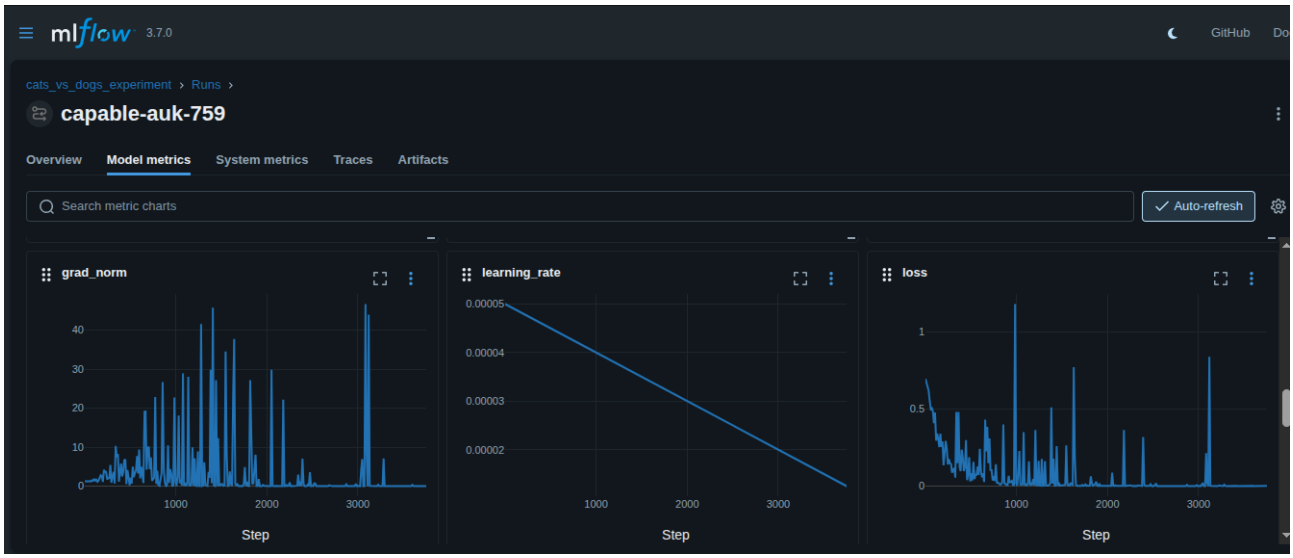
Dentro del experimento se visualizan cada una de las corridas ejecutadas:



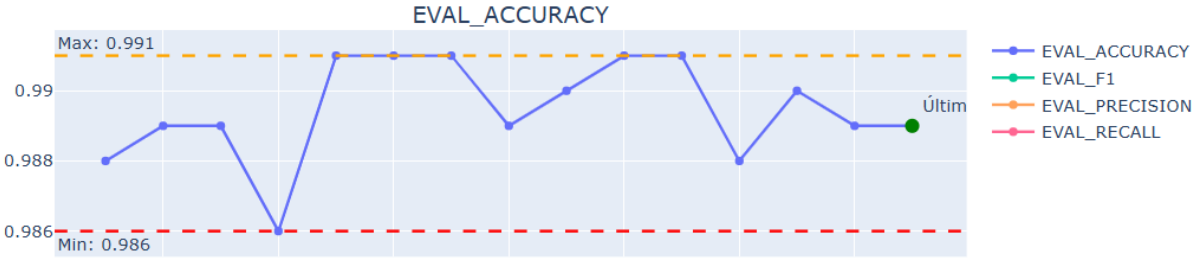
Y a su vez podemos ver en detalle todas las métricas generadas durante el entrenamiento:

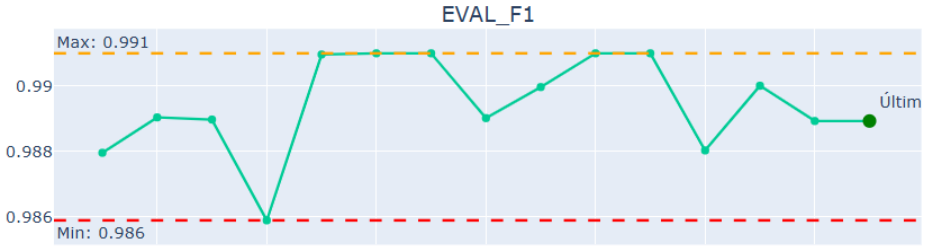


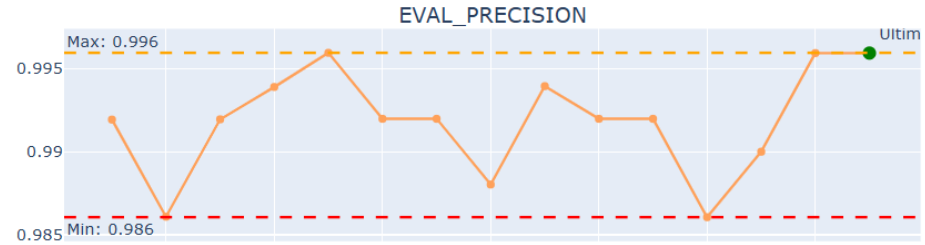


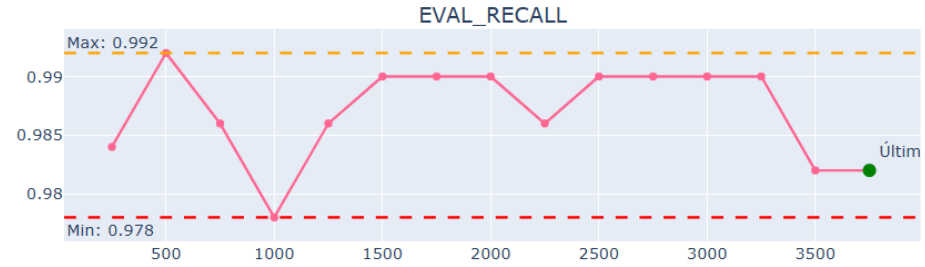


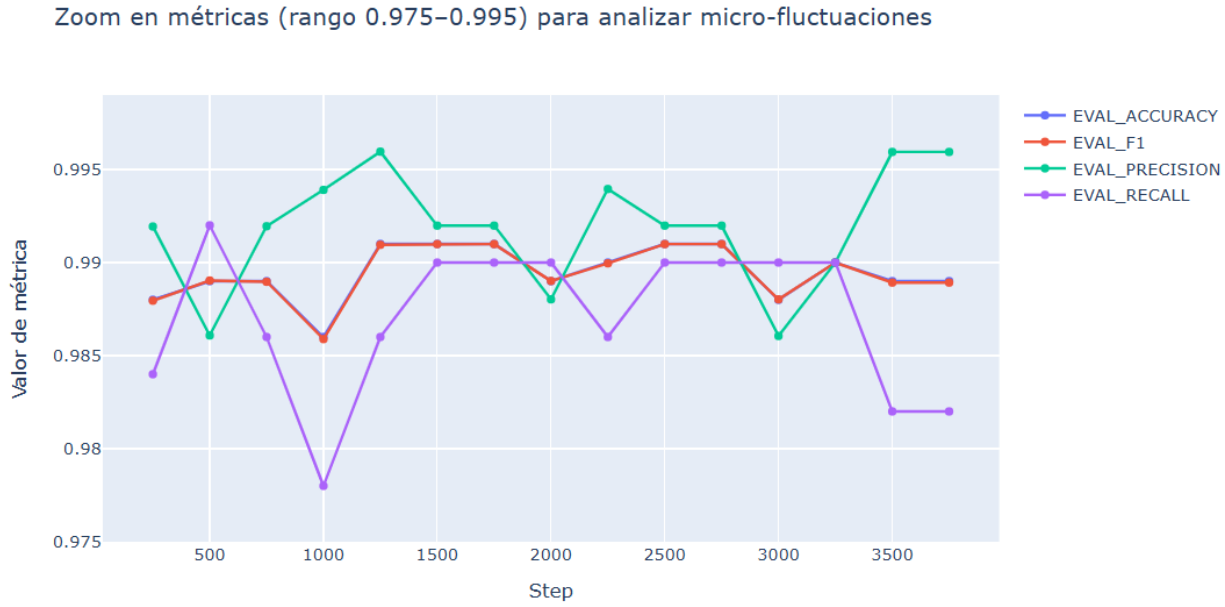
Asímismo, tomando los archivos de métricas registrados en mlruns como inputs, generamos nuestro propio dashboard interactivo de métricas para evaluar la progresión (**Metrics.ipynb**):




EVAL_F1
Max: 0.991
0.99
0.988
0.986
Min: 0.986
Últim


EVAL_PRECISION
Max: 0.996
0.995
0.99
0.985
Min: 0.986
Últim


EVAL_RECALL
Max: 0.992
0.99
0.985
0.98
Min: 0.978
Últim
500
1000
1500
2000
2500
3000
3500


Zoom en métricas (rango 0.975–0.995) para analizar micro-fluctuaciones
EVAL_ACCURACY
EVAL_F1
EVAL_PRECISION
EVAL_RECALL
Valor de métrica
0.995
0.99
0.985
0.98
0.975
500
1000
1500
2000
2500
3000
3500
Step

El comportamiento que se observa es de métricas de clasificación **altamente precisas,** ya que si bien los valores fluctúan, siempre se mantienen en un rango acotado entre 0.97-0.99.

- ✓ **Alta performance del modelo**
  - Está clasificando correctamente la mayor parte de las imágenes.
  - Prácticamente todos los gatos y perros se están prediciendo correctamente.

- ✓ **Pequeñas fluctuaciones**
  - Las subidas y bajadas reflejan **variaciones naturales entre batches o steps** de evaluación.
  - Por ejemplo, en algunos pasos el modelo podría predecir 1–2 imágenes mal, lo que hace que la métrica baje ligeramente, y luego vuelve a subir cuando las siguientes predicciones son correctas.

- ✓ **Rango estrecho indica estabilidad**
  - Que todas las métricas estén siempre en 0.97/0.98/0.99 muestra que el modelo **está entrenado y generaliza muy bien**.
  - Si hubiera picos muy grandes (por ejemplo, de 0.80 a 0.99) sería señal de **inestabilidad** / overfitting.

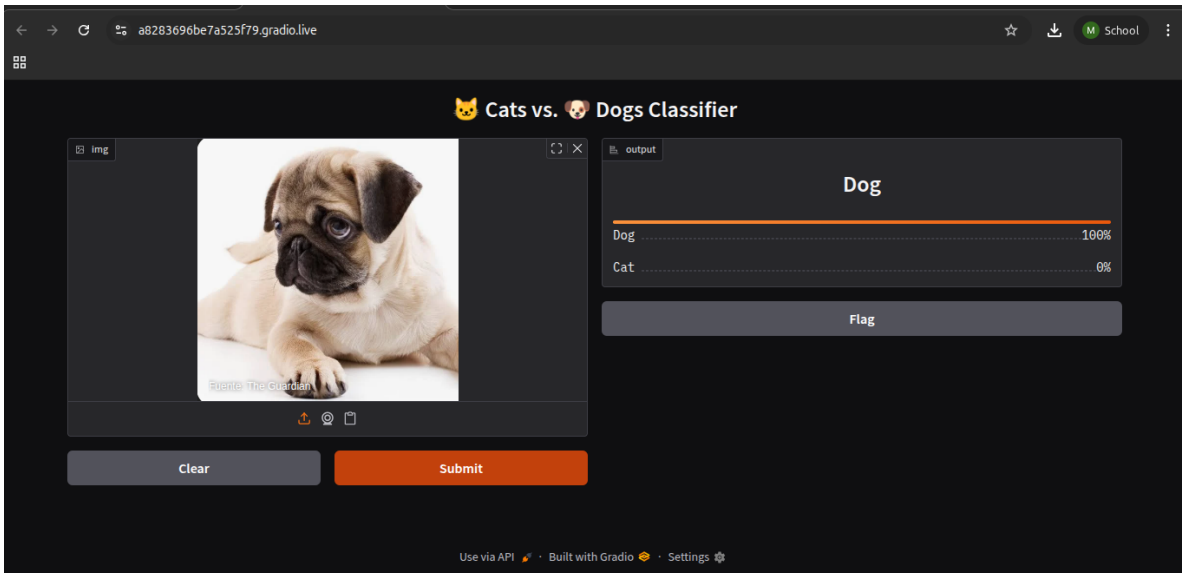
Resultados

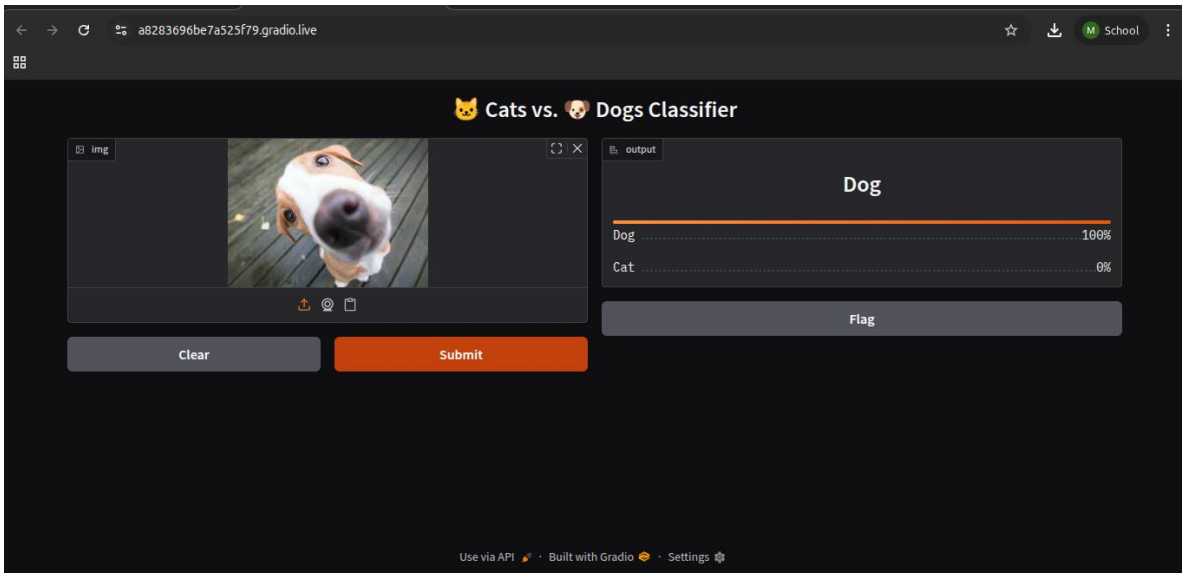
Se probó el modelo con la app de inferencia dando muy buenos resultados.
Algunas de las imágenes seleccionadas y las evidencias de prueba se encuentran en la carpeta **tests/**

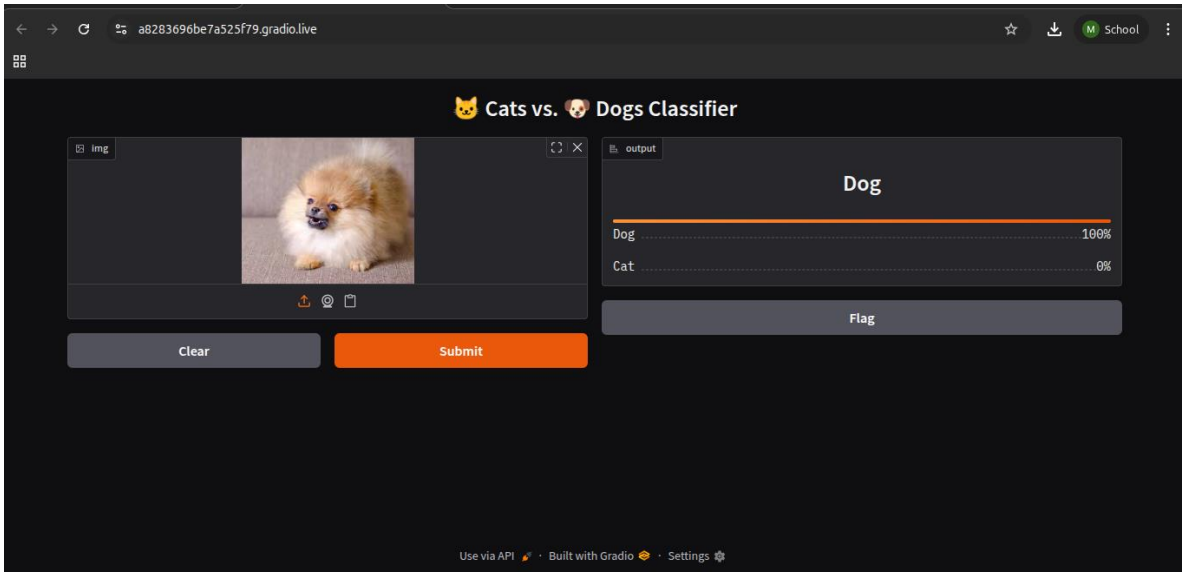
Observaciones:

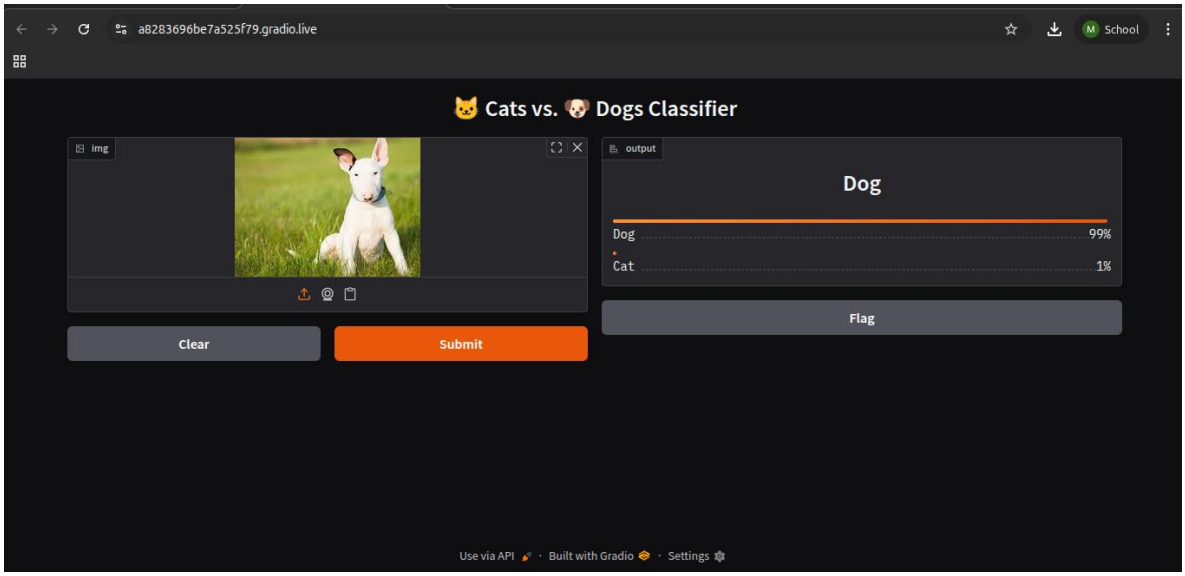
- Los perros de diferentes razas, aún las muy exóticas, fueron predichos correctamente.

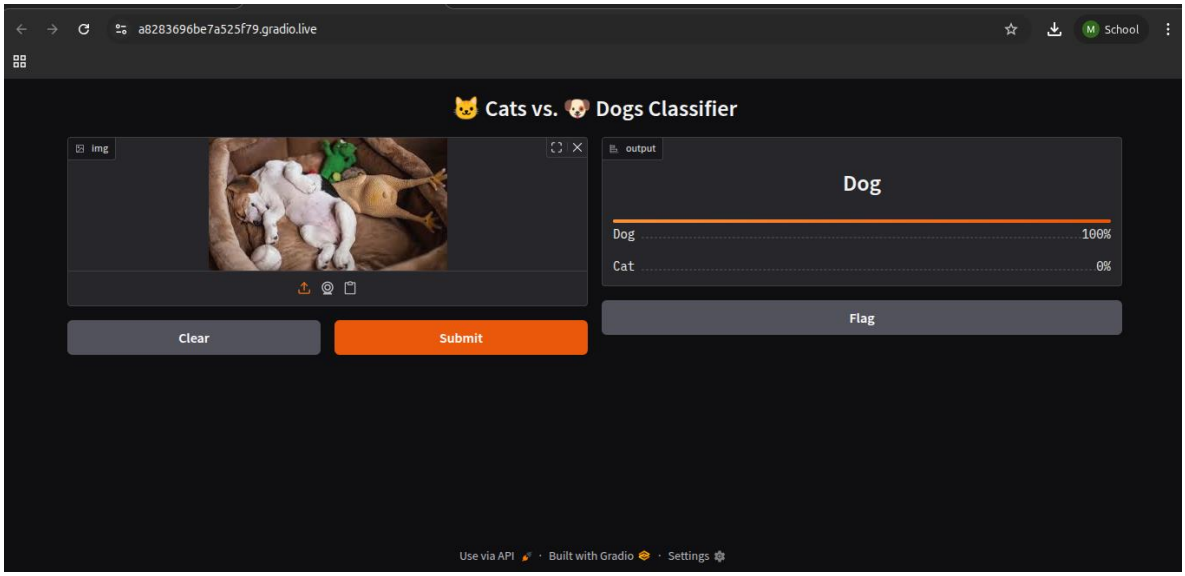
Ejemplos:





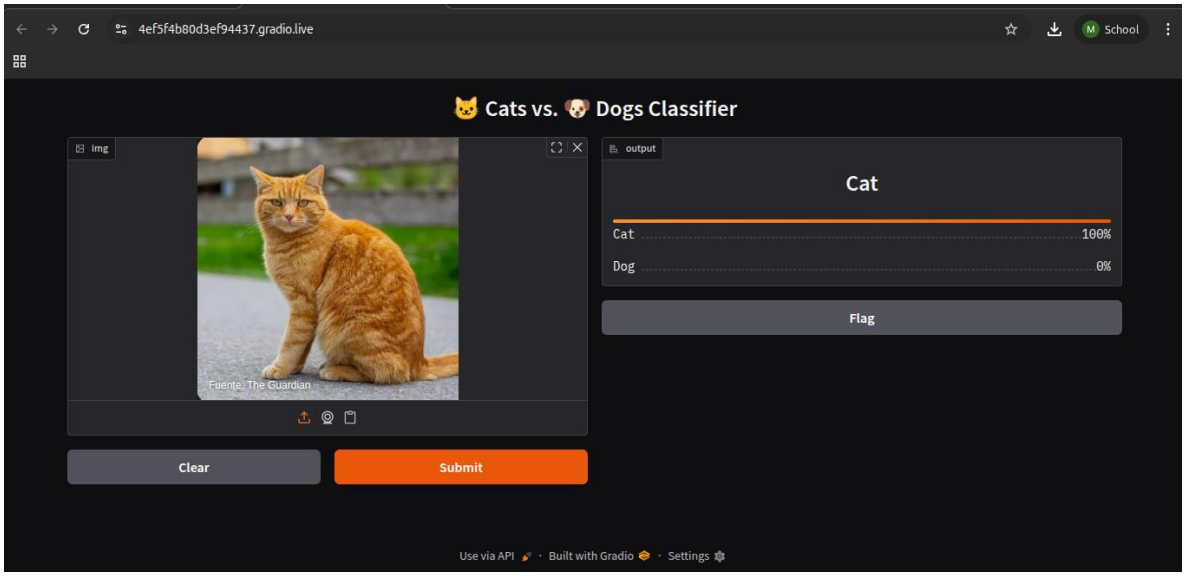

a8283696be7a525f79.gradio.live
School
Cats vs. Dogs Classifier
img
output
Dog
Dog 100%
Cat 0%
Flag
Clear
Submit
Use via API · Built with Gradio · Settings

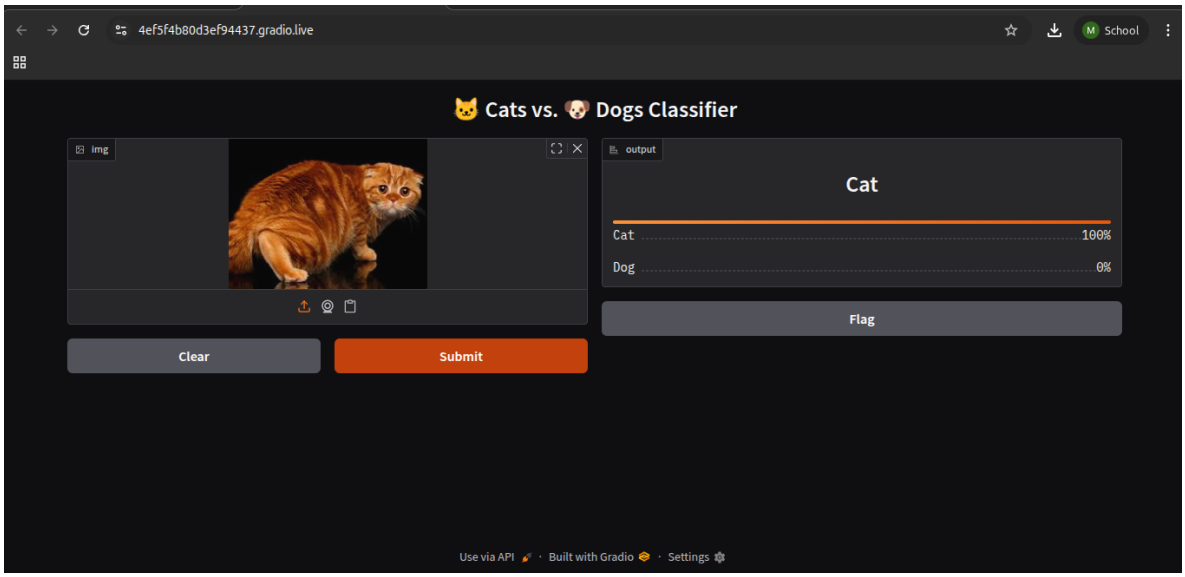

a8283696be7a525f79.gradio.live
School
Cats vs. Dogs Classifier
img
output
Dog
Dog 99%
Cat 1%
Flag
Clear
Submit
Use via API · Built with Gradio · Settings

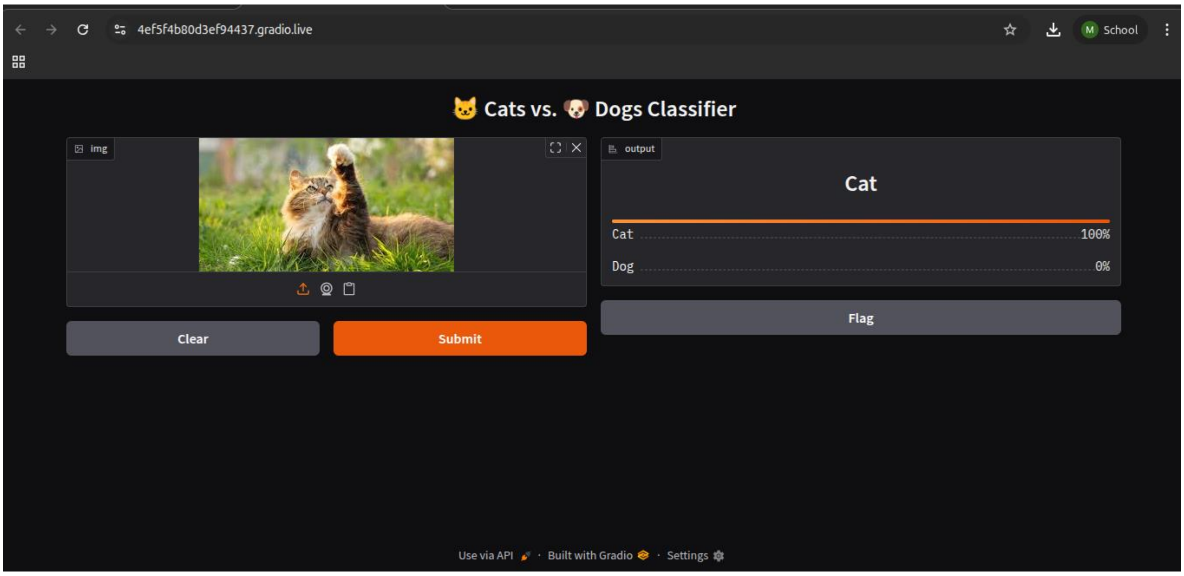

a8283696be7a525f79.gradio.live
School
Cats vs. Dogs Classifier
img
output
Dog
Dog 100%
Cat 0%
Flag
Clear
Submit
Use via API · Built with Gradio · Settings

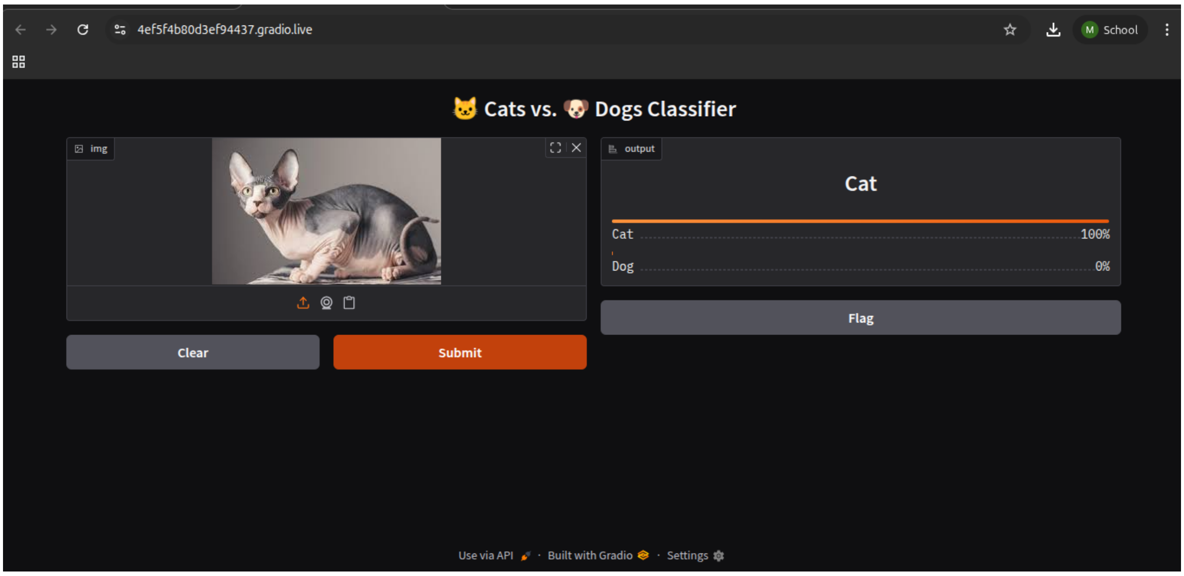
- Los gatos de diferentes razas, fueron predichos correctamente, excepto en un caso, donde el gato está acostado sobre un oso de peluche.

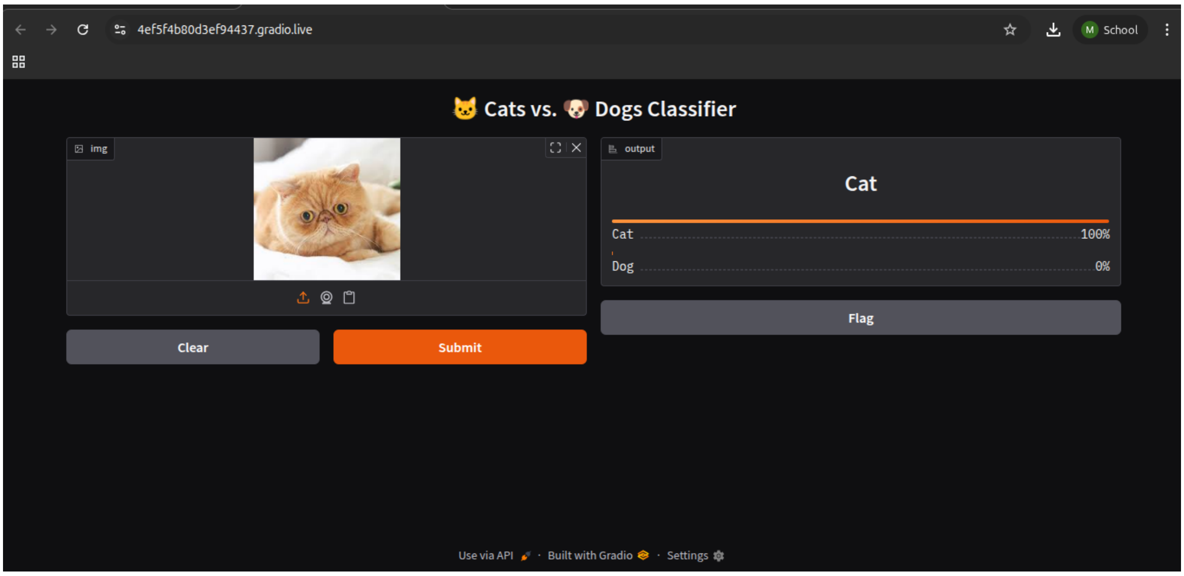
Ejemplos:



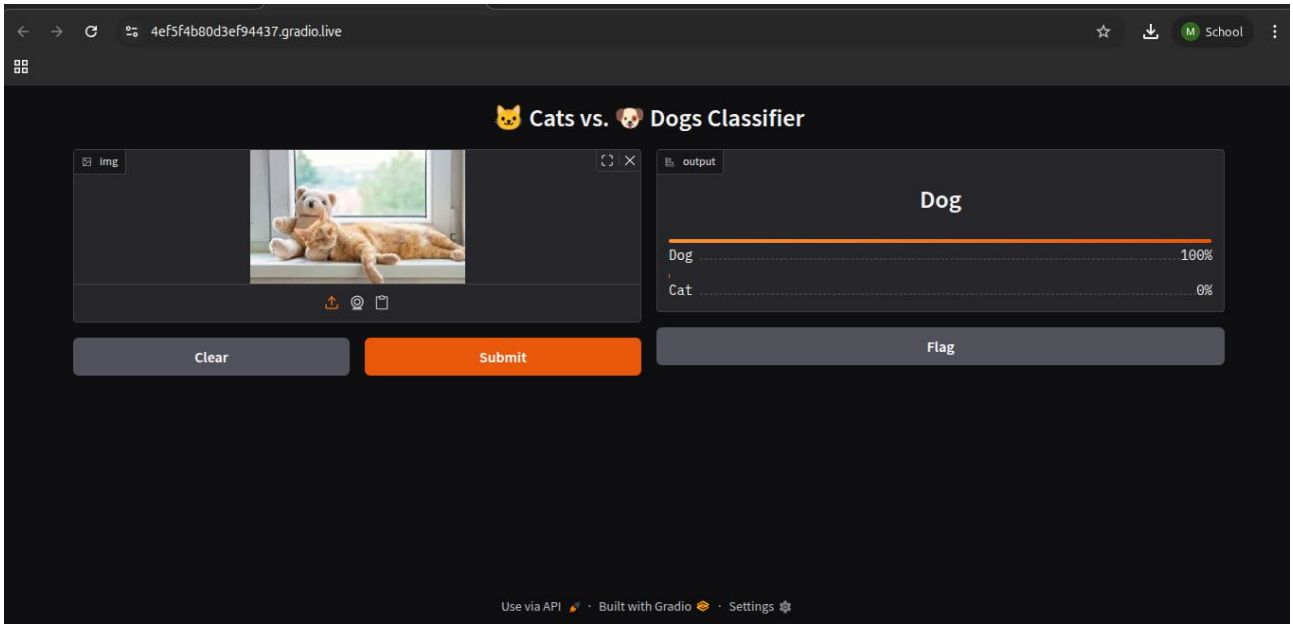



4ef5f4b80d3ef94437.gradio.live
School
Cats vs. Dogs Classifier
img
output
Cat
Cat 100%
Dog 0%
Clear
Submit
Flag
Use via API · Built with Gradio · Settings


4ef5f4b80d3ef94437.gradio.live
School
Cats vs. Dogs Classifier
img
output
Cat
Cat 100%
Dog 0%
Clear
Submit
Flag
Use via API · Built with Gradio · Settings

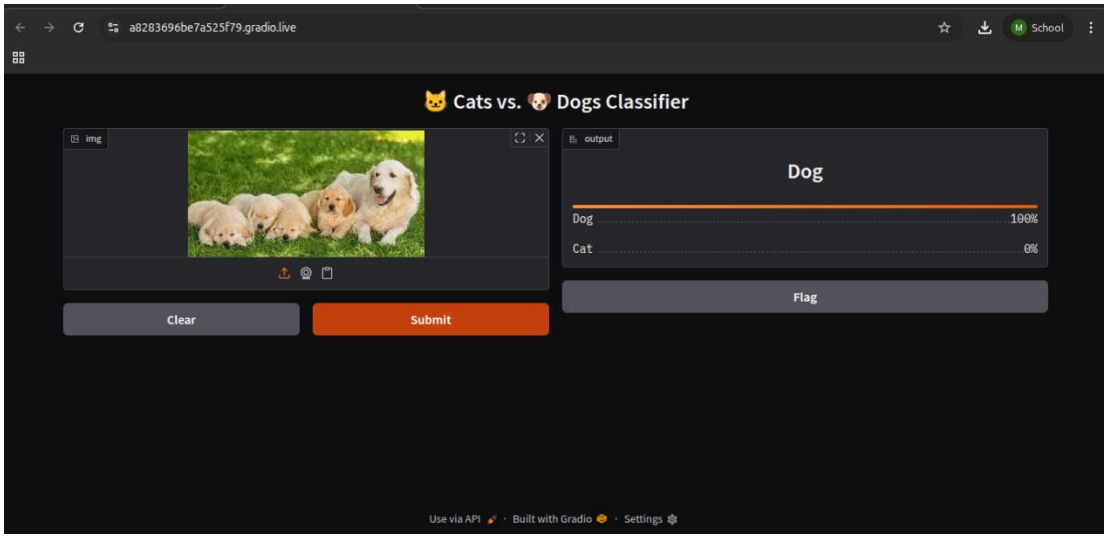

4ef5f4b80d3ef94437.gradio.live
School
Cats vs. Dogs Classifier
img
output
Cat
Cat 100%
Dog 0%
Clear
Submit
Flag
Use via API · Built with Gradio · Settings

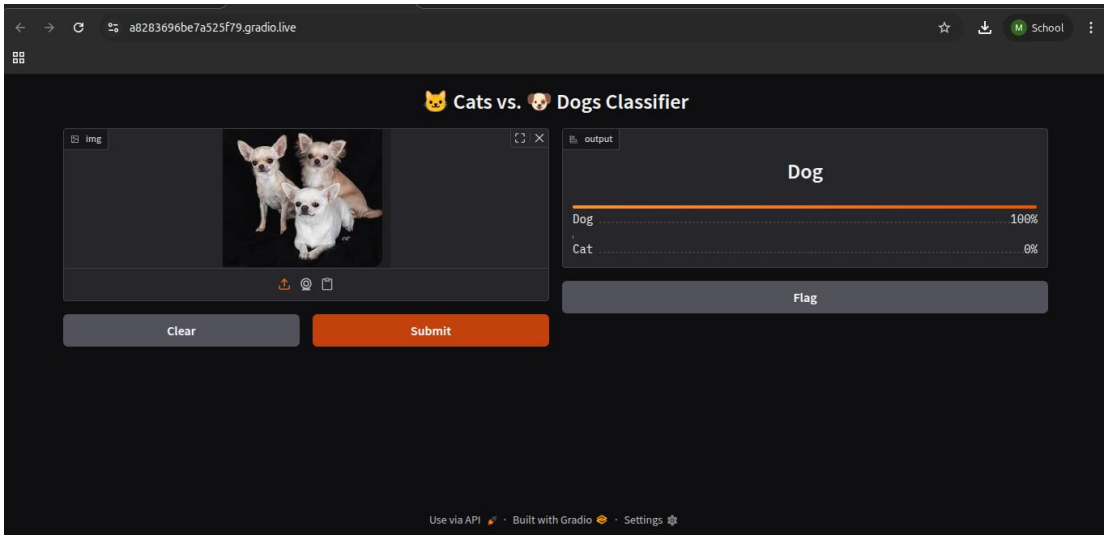
✗ Error de clasificación



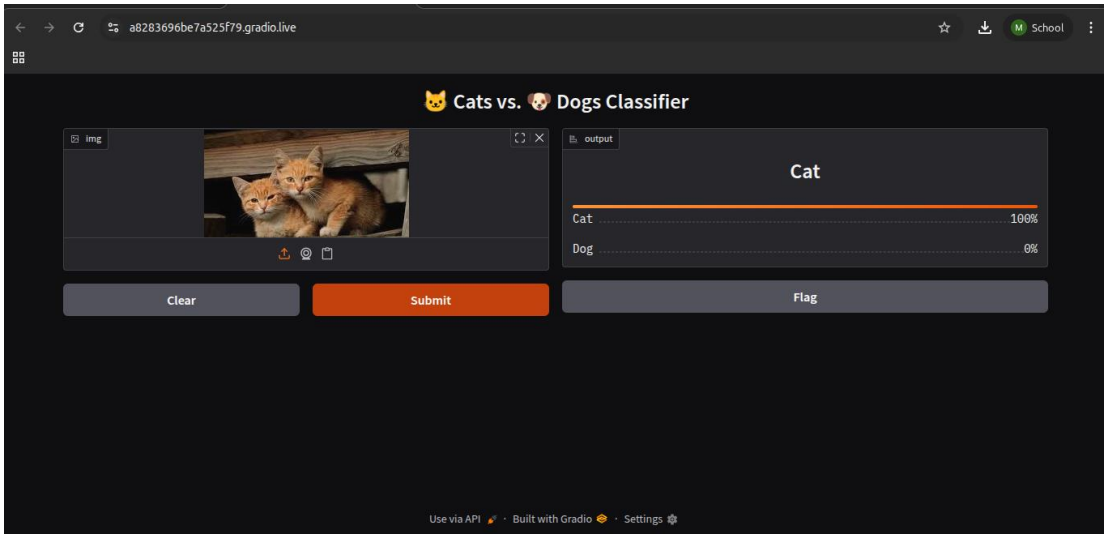
Para desafiar al modelo se probaron otros escenarios para ver los resultados.

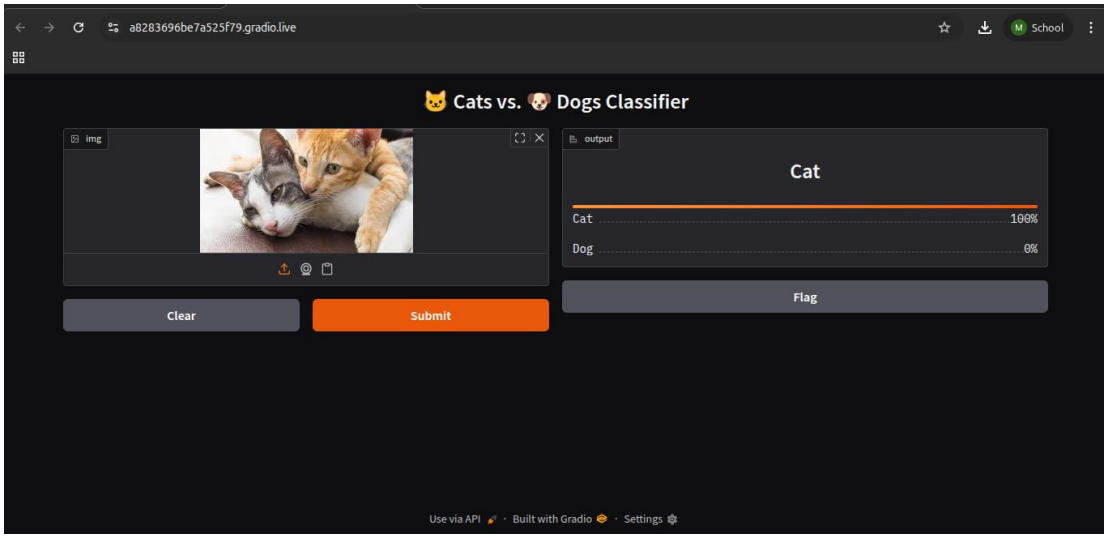
1) Más de un ejemplar en la foto.
   a. Más de un perro: predijo correctamente. Ejemplos:



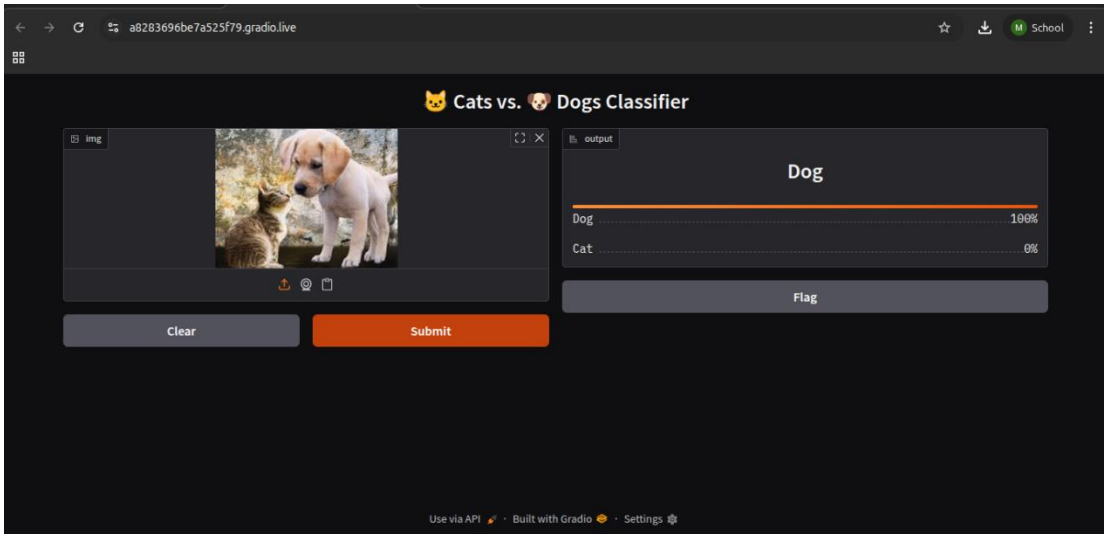


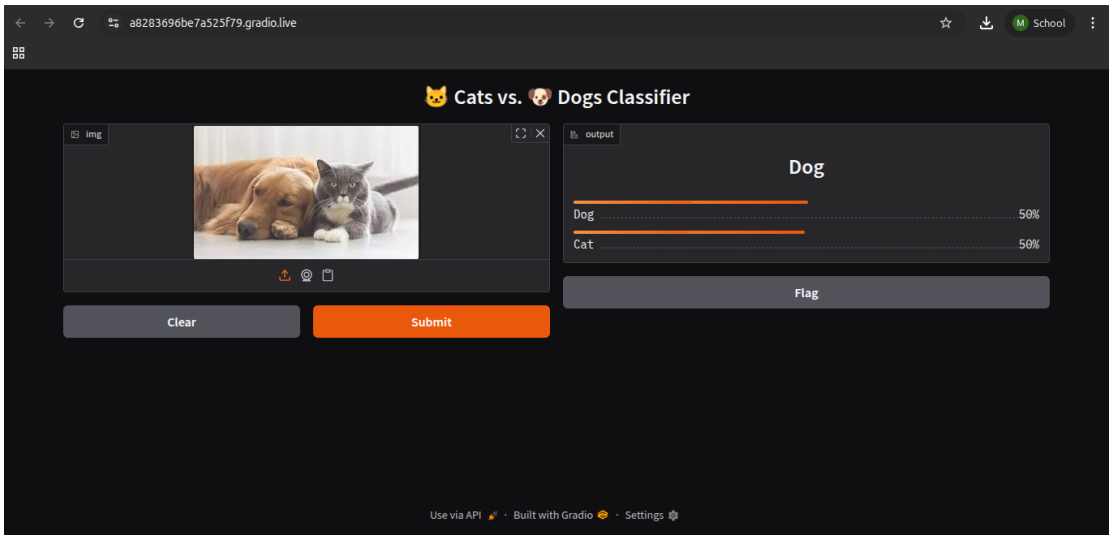
b. Más de un gato: también predijo correctamente. Ejemplos:

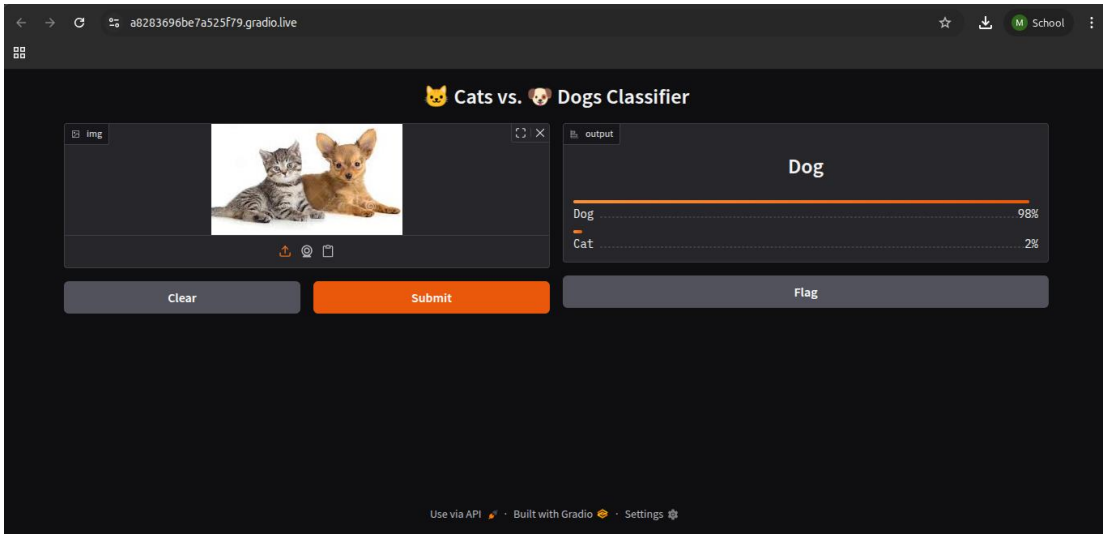




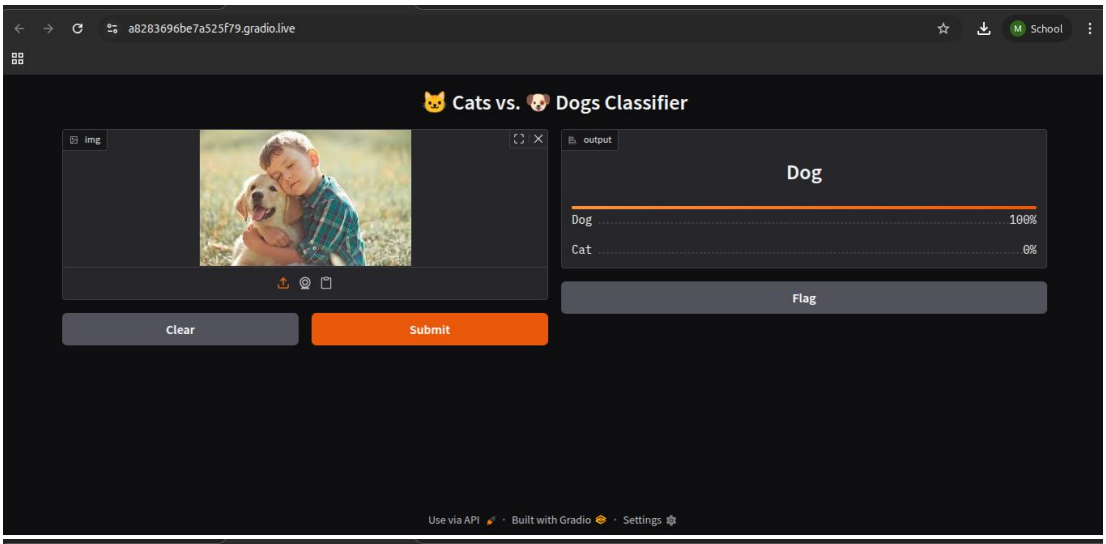
C. Perro y gato en la misma foto: predijo mayormente perro como clase positiva, excepto en un par de casos en los que se redujo el porcentaje de probabilidad. Ejemplos:

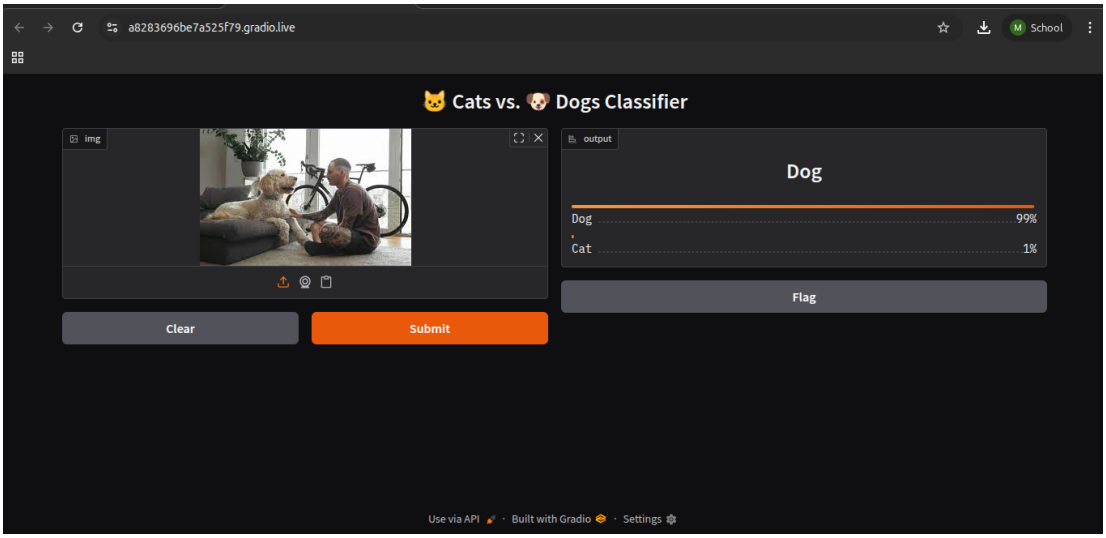


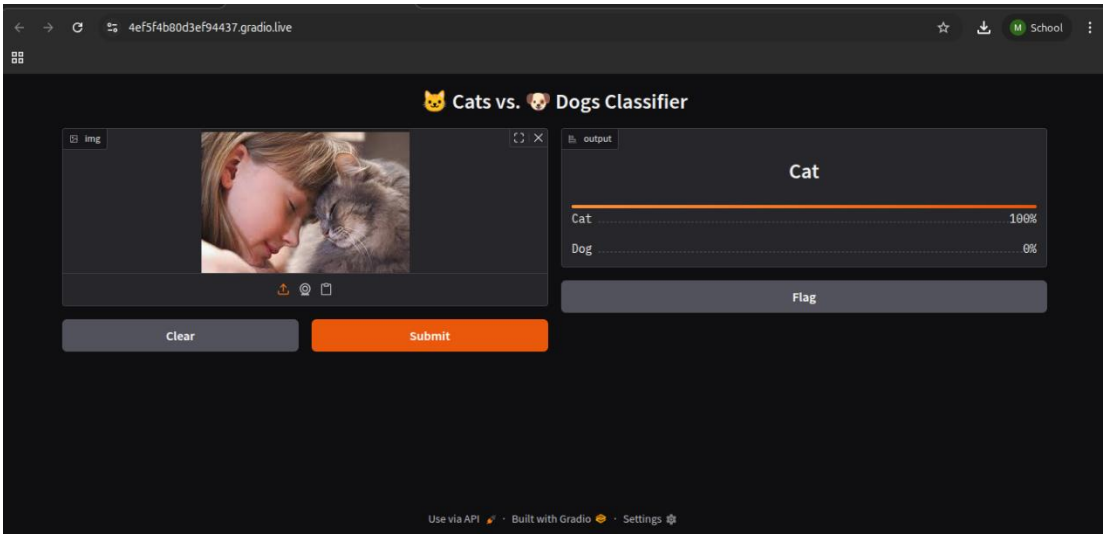




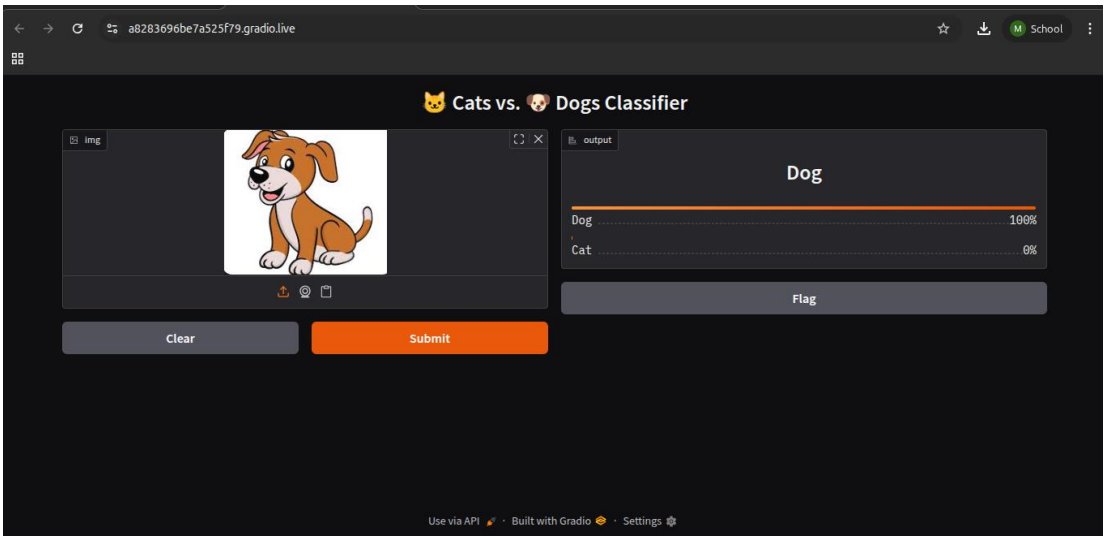
2) La foto incluye además de la mascota a una persona: predijo correctamente. Ejemplos:

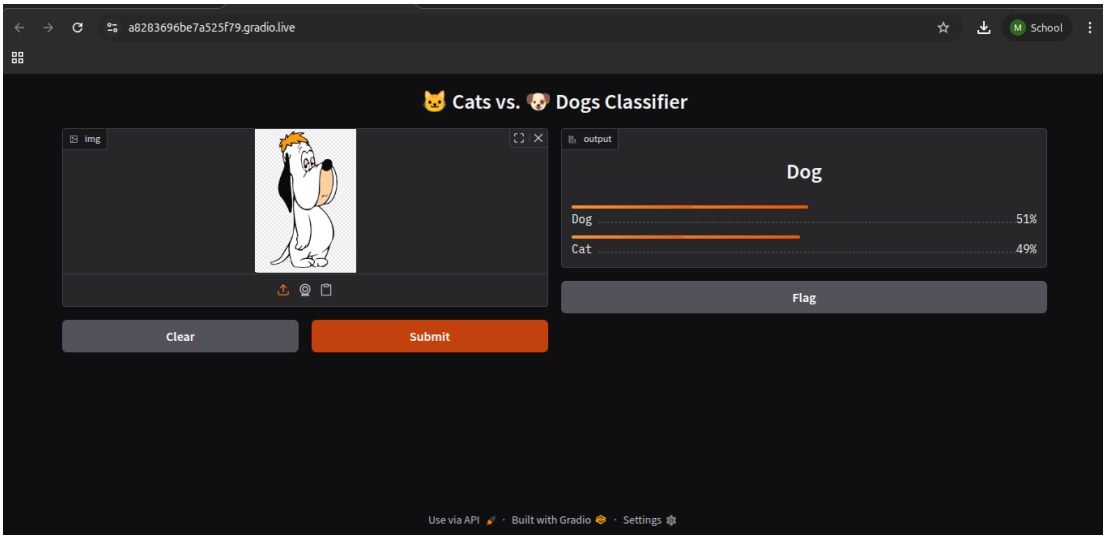


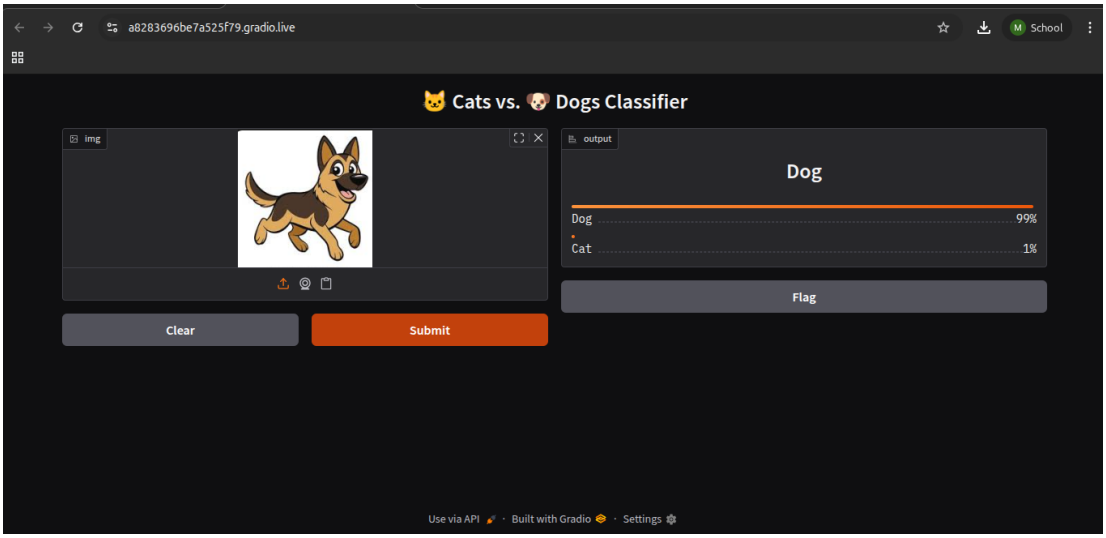


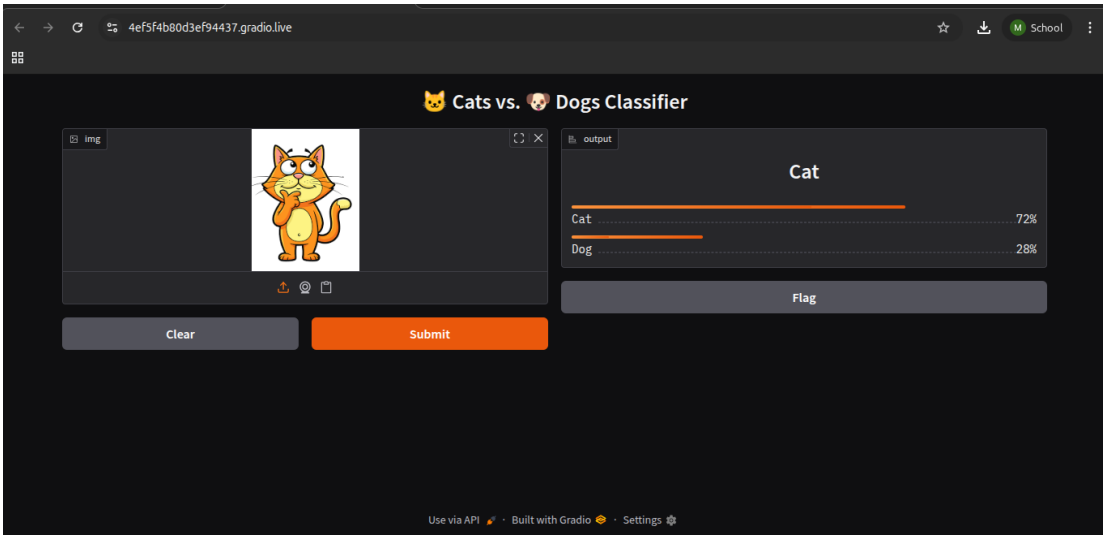


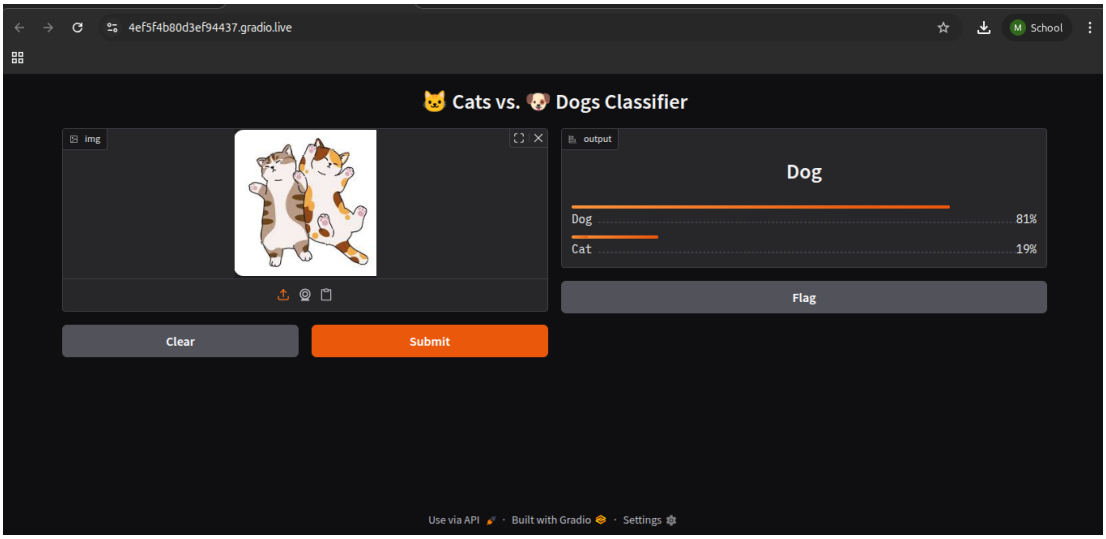
3) La foto no es un animal real, si no, un dibujo: en ese caso, la mayor cantidad de las veces arroja “perro”, demostrando que no evalúa correctamente este tipo de imágenes. Esto es esperable, ya que fue entrenado con fotos de animales reales. Ejemplos:

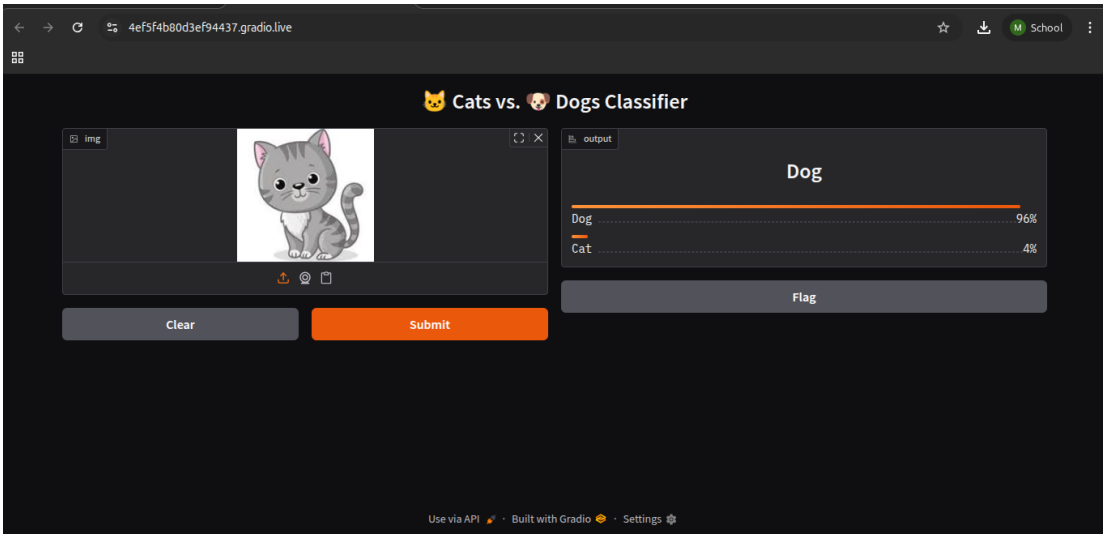




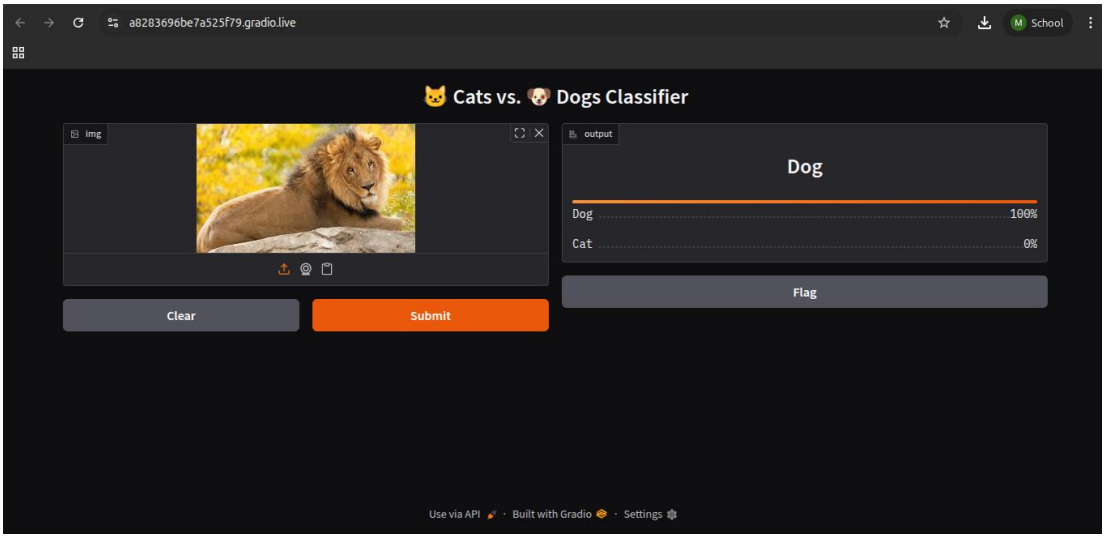


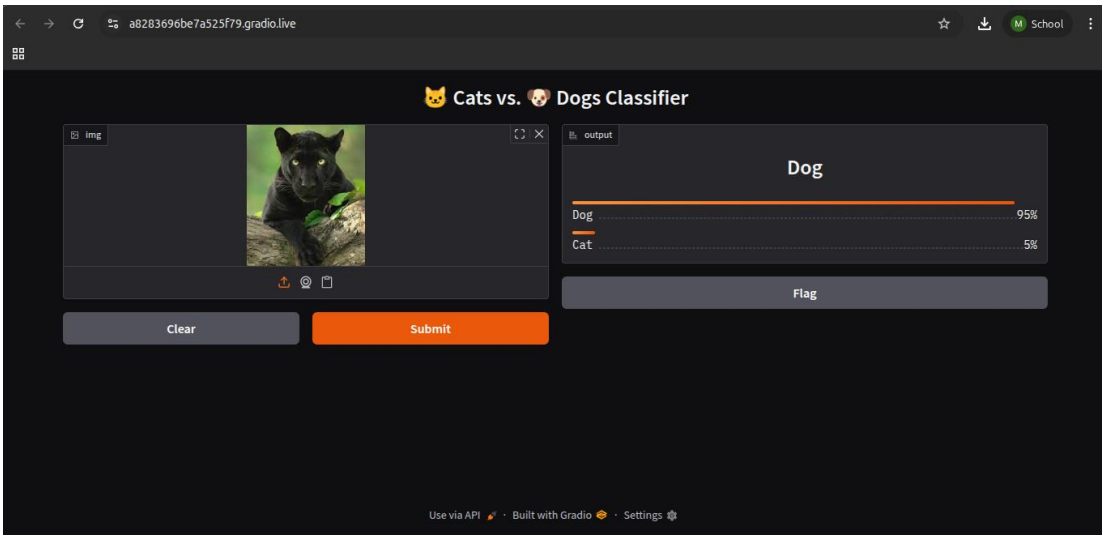

4ef5f4b80d3ef94437.gradio.live
School
Cats vs. Dogs Classifier
img
output
Cat
Cat 72%
Dog 28%
Flag
Clear
Submit
Use via API · Built with Gradio · Settings

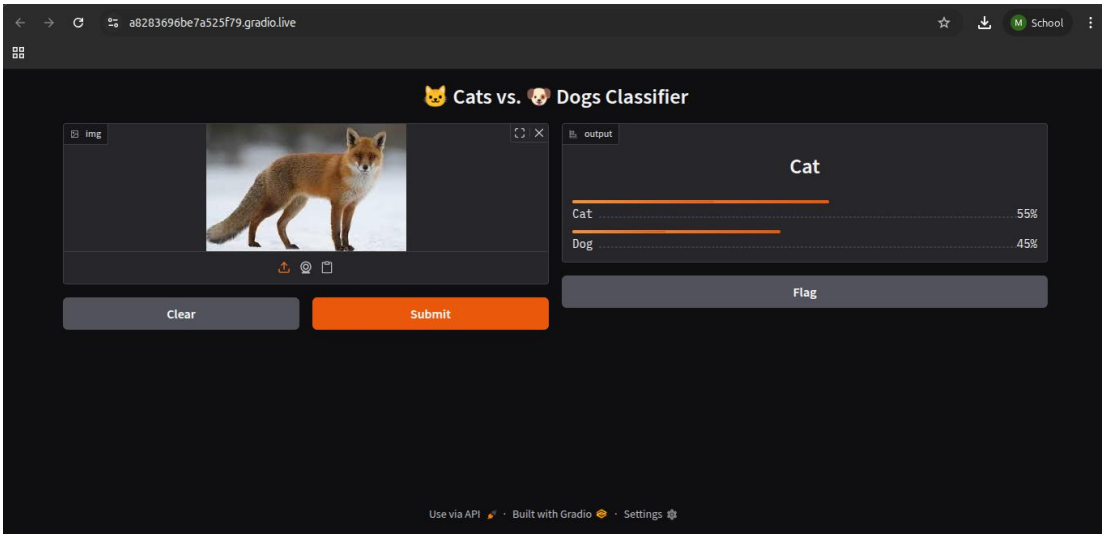

4ef5f4b80d3ef94437.gradio.live
School
Cats vs. Dogs Classifier
img
output
Dog
Dog 81%
Cat 19%
Flag
Clear
Submit
Use via API · Built with Gradio · Settings


4ef5f4b80d3ef94437.gradio.live
School
Cats vs. Dogs Classifier
img
output
Dog
Dog 96%
Cat 4%
Flag
Clear
Submit
Use via API · Built with Gradio · Settings

4) La foto no es de un gato o perro, si no, de otro animal: siempre la mayor probabilidad arroja “perro”, aún en felinos como león, pantera o tigre. A pesar de que el zorro pertenece a la familia de los cánidos, lo predijo incorrectamente como gato. Ejemplos:







Conclusiones

Este proyecto representa una **implementación práctica y concreta** de fine-tuning de un modelo de visión por computadora moderno, **MobileViT**, aplicado al clásico problema de clasificación binaria — distinguir entre imágenes de gatos y perros — mediante el dataset "**Cats vs Dogs**" de **Hugging Face.**

Gracias a este enfoque, el proyecto demuestra que modelos compactos y eficientes pueden adaptarse con relativamente pocos recursos a tareas específicas (**transfer learning**), logrando resultados de alta performance en un dominio concreto. Esto permite:

- aprovechar las representaciones aprendidas en grandes datasets preentrenados (generalización de bajo nivel: bordes, texturas, patrones)
- entrenar eficientemente sobre un dataset pequeño o moderado, acortando tiempos y requerimientos de hardware
- ofrecer una solución movilizable — tanto para su uso local, como mediante una interfaz web de inferencia en tiempo real.

Aunque el proyecto ya cumple bien su objetivo inicial, hay múltiples formas de expandirlo, robustecerlo o adaptarlo a nuevos escenarios. Algunas posibilidades:

Área de mejora	Propuesta	Beneficio / Impacto
Dataset	Migrar a un dataset más complejo o multiclase.	Permite evaluar la verdadera capacidad del modelo en escenarios reales, reduce overfitting, da relevancia práctica y escala el proyecto más allá del caso simple binario.
Dataset	Aumentar el tamaño y diversidad del dataset mediante data augmentation avanzada.	Mejora la generalización del modelo permitiendo atacar los casos en que el modelo actual falla, fundamental para despliegues robustos.
Arquitectura del modelo	Probar otras variantes de MobileViT.	Incremento de precisión y eficiencia sin aumentar significativamente el costo computacional.
Arquitectura del modelo	Explorar otras arquitecturas.	Comparación científica entre enfoques; permite seleccionar el mejor trade-off para el caso de uso.
Pipeline de entrenamiento y evaluación	Estandarizar y automatizar el pipeline en una estructura modular tipo "end-to-end ML pipeline"	Facilita reentrenamientos, CI/CD en ML, experimentación guiada y mantenibilidad del proyecto.
Escalabilidad y experimentación	Crear plantillas de experimentación automática, por ejemplo, para la búsqueda de hiperparámetros.	Permite encontrar la configuración óptima sin intervención manual; mejora resultados sin esfuerzo humano.
Optimización para producción	Optimizar el modelo para inferencia, exportar y disponibilizar app web real / API REST.	Permite despliegue en distintos sistemas, servidores o dispositivos, y puede conectar al modelo con aplicaciones reales, facilitando la integración con otras apps, dashboards o sistemas externos.

En resumen, si bien el proyecto no tiene calidad productiva, si no que fue pensado dentro del ámbito académico, los resultados no sólo demuestran lo que se puede lograr con pocos recursos y un dataset moderado usando transferencia de aprendizaje — también deja abierta la puerta a muchos caminos de crecimiento, especialización y aplicación práctica.

Anexos

GitHub

Repositorio GitHub con todo el contenido del trabajo.

Referencias

Material de clase "Visión por computadora 3"
https://github.com/FIUBA-Posgrado-Inteligencia-Artificial/CEIA-ViT

Hugging Face

https://huggingface.co/

mlflow

https://mlflow.org/