

Sistema de Clasificación Automática de Edificios Urbanos mediante Deep Learning

1. Problema que abordamos y su contexto

En el contexto de las ciudades inteligentes, la planificación urbana, el análisis inmobiliario y los sistemas de información geográfica, existe una necesidad creciente de clasificar automáticamente estructuras urbanas a partir de imágenes. Actualmente, la categorización de edificaciones (por ejemplo, viviendas, edificios comerciales, iglesias o estructuras industriales) se realiza de forma manual o semi-automática, lo cual resulta costoso, lento y difícil de escalar.

El avance reciente de la visión por computador y el *deep learning* ha demostrado un alto potencial para automatizar tareas de clasificación visual. Sin embargo, muchos sistemas existentes requieren datos altamente curados o capturados en condiciones controladas. En este proyecto abordamos el problema de la clasificación automática de edificaciones urbanas a partir de imágenes reales de calles, utilizando redes neuronales convolucionales entrenadas con datos públicos.

Nuestro trabajo se sitúa en un escenario realista, con imágenes heterogéneas capturadas desde distintos ángulos, condiciones de iluminación y contextos urbanos, lo que incrementa la relevancia práctica de la solución propuesta.

2. Pregunta de negocio y alcance del proyecto

Pregunta de negocio

¿Es posible desarrollar un sistema basado en *deep learning* capaz de clasificar automáticamente distintos tipos de edificaciones urbanas a partir de imágenes de calles, con un nivel de precisión suficiente para apoyar procesos de análisis urbano, distribución logística y aplicaciones académicas?

Objetivo general

Desarrollar y evaluar un sistema de clasificación de imágenes que identifique automáticamente el tipo de edificación urbana presente en una imagen de calle.

Alcance del proyecto

- Entrenamiento de un modelo de clasificación supervisada basado en redes neuronales convolucionales.
- Uso de *transfer learning* a partir de modelos preentrenados en ImageNet.
- Clasificación de ocho categorías de edificaciones urbanas.
- Evaluación del desempeño mediante métricas estándar de clasificación.

Fuera de alcance

- Detección de múltiples edificaciones en una sola imagen (el proyecto se centra en clasificación, no en *object detection*).
 - Segmentación semántica.
 - Recolección de datos propios.
 - Despliegue productivo a gran escala.
-

3. Conjunto de datos a emplear

Utilizamos el conjunto de datos público **House Rooms & Streets Image Dataset**, disponible en Kaggle y publicado por el usuario *mikhailma*. Para este proyecto empleamos específicamente la carpeta **street**, la cual contiene imágenes de edificaciones urbanas.

Categorías consideradas (8 clases):

1. Apartment buildings (apartamentos)
2. Churches (iglesias)
3. Garages (garajes)
4. Houses (casas)
5. Industrial buildings (edificios industriales)
6. Office buildings (edificios de oficinas)
7. Retail buildings (comercios)
8. Roofs (techos)

Características técnicas del dataset:

- Tipo de datos: imágenes RGB de exteriores
- Formato: JPG / PNG
- Número aproximado de imágenes: 20.000
- Resolución variable
- Volumen estimado: 175MB
- Licencia: CC0 – Dominio Público

El uso exclusivo del subconjunto street permite enfocar el proyecto en el análisis y reconocimiento de estructuras urbanas, evitando introducir variabilidad adicional relacionada con espacios interiores que no resulta relevante para los objetivos planteados. Al tratarse de un problema de clasificación multiclase con ocho categorías, el modelo debe aprender a distinguir entre edificaciones que, en algunos casos, comparten características visuales similares.

Las imágenes presentan una amplia variabilidad en cuanto a iluminación, ángulo de captura, tamaño de las edificaciones y elementos presentes en la escena. Esta diversidad refleja condiciones cercanas a escenarios reales, lo cual permite entrenar modelos de visión por computador que no dependan de patrones demasiado específicos del conjunto de entrenamiento. Un aspecto importante del conjunto de datos es la variabilidad dentro de cada clase, así mismo, algunas clases presentan fronteras visuales poco definidas, como ocurre entre edificios de oficinas, comercios y apartamentos. Esta situación es útil para analizar el comportamiento del modelo y comprender mejor sus limitaciones. Además, las imágenes incluyen elementos propios de entornos reales, como vehículos, personas u objetos que no forman parte directa de la edificación, lo que introduce cierto nivel de ruido visual. El dataset se encuentra organizado en carpetas por clase, lo que facilita su uso directo en procesos de entrenamiento con modelos de aprendizaje profundo.

Los datos son descargados y preparados una única vez al inicio del proyecto. No se considera la incorporación de nuevos datos durante el desarrollo del estudio, con el fin de mantener estabilidad experimental y asegurar la reproducibilidad de los resultados obtenidos.

3.1 Creación del Dataset

El proyecto incluye la preparación del conjunto de datos, el cual se implementa en uno de los scripts del repositorio https://github.com/marlon-alvarez/clasificador_edificios.

Dicho procedimiento tiene como objetivo transformar las imágenes originales en una representación estructurada que pueda ser utilizada de forma eficiente durante el entrenamiento del modelo.

Dado que las imágenes por sí solas no constituyen un formato directamente utilizable por los algoritmos de aprendizaje automático, fue necesario organizar la información en una estructura más adecuada. Para ello, se generó un dataset en forma de tabla que relaciona cada imagen con su correspondiente categoría. Esta tabla está compuesta por dos atributos: uno que referencia la ubicación de la imagen dentro del directorio del proyecto y otro que indica la clase a la que pertenece, como se observa en la imagen 1.

Esta estrategia de organización permite una gestión más controlada de los datos, facilita su carga durante las distintas etapas del experimento y contribuye a mantener la consistencia entre las imágenes y sus etiquetas a lo largo del proceso de entrenamiento y evaluación del modelo.

```

import os
import pandas as pd
from tqdm import tqdm

root = '/kaggle/input/house-rooms-streets-image-dataset/kaggle_room_street_data/street_data'

img_paths = []
labels = []

for file in tqdm(os.listdir(root), desc='street_data'):
    img_paths.append(os.path.join(root, file))
    label = file.split('_')[0]
    labels.append(label)

df = pd.DataFrame({
    'path': img_paths,
    'label': labels
})

print(df.head())
print('Total imágenes:', len(df))

street_data: 100%|██████████| 19658/19658 [00:00<00:00, 560563.67it/s]
          path      label
0  /kaggle/input/house-rooms-streets-image-dataset/kaggle_room_street_data/street_data/roo...
1  /kaggle/input/house-rooms-streets-image-dataset/kaggle_room_street_data/street_data/garage...
2  /kaggle/input/house-rooms-streets-image-dataset/kaggle_room_street_data/street_data/churc...
3  /kaggle/input/house-rooms-streets-image-dataset/kaggle_room_street_data/street_data/churc...
4  /kaggle/input/house-rooms-streets-image-dataset/kaggle_room_street_data/street_data/industri...
Total imágenes: 19658

```

Imagen 1. Preparación de ubicación y etiqueta por imagen

4. Repositorio Github para el código

El código del proyecto se gestiona mediante un repositorio git centralizado, el cual contiene:

- Scripts de preprocesamiento de datos.
- Notebooks de exploración y experimentación.
- Código de entrenamiento y evaluación de modelos.
- Documentación técnica del proyecto.

Repositorio Github:

https://github.com/marlon-alvarez/clasificador_edificios

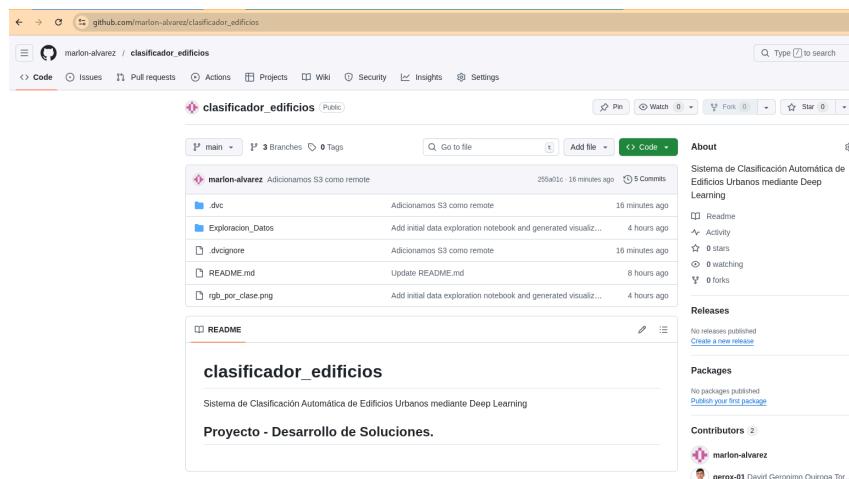


Imagen 2. Repositorio creado en github

El control de versiones se realiza siguiendo buenas prácticas, con commits descriptivos y revisión colaborativa del código antes de integrar cambios en la rama principal. Como ejemplo se incluye imagen del PR para la maqueta o prototipo del proyecto.

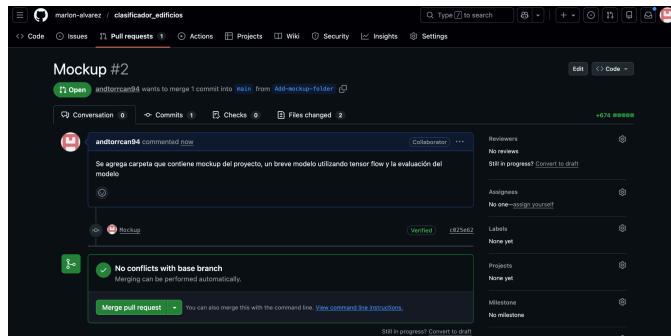


Imagen 3. Pull request al repositorio para agregar cambios.

5. Repositorio DVC para los datos

Para el control de versiones de los datos y artefactos pesados del proyecto, utilizamos **DVC** (**Data Version Control**) con un almacenamiento remoto en Amazon S3.

Estrategia adoptada:

- Los datos originales descargados desde Kaggle se almacenan como datos *raw*.
- Los datos procesados (imágenes redimensionadas, normalizadas y divididas en *train/validation/test*) se versionan mediante DVC.
- Los modelos entrenados y resultados relevantes también se incluyen en el versionado.

Esta estrategia nos permite:

- Reproducir experimentos de forma consistente.
- Evitar el almacenamiento de archivos pesados en el repositorio Git.
- Mantener trazabilidad completa entre código, datos y resultados.

Bucket S3

The screenshot shows the Amazon S3 console interface. On the left, there's a navigation sidebar with sections like 'Amazon S3', 'Buckets', 'Seguridad y administración de acceso', 'Información y administración de almacenamiento', and 'AWS Marketplace para S3'. The main area is titled 'md5/' and shows a list of 'Objetos (256)'. The table lists items with the following details:

Nombre	Tipo	Última modificación	Tamaño
00/	Carpeta	-	
01/	Carpeta	-	
02/	Carpeta	-	
03/	Carpeta	-	
04/	Carpeta	-	
05/	Carpeta	-	
06/	Carpeta	-	
07/	Carpeta	-	
08/	Carpeta	-	
09/	Carpeta	-	
0a/	Carpeta	-	
0b/	Carpeta	-	

Imagen 4. Bucket S3 creado y con datos enviados desde la instancia EC2

Envío de datos al bucket s3

```
ec2-user@ip-172-31-14-175:~/proyecto/clasificador_edificios$ dvc add datos/
/home/ec2-user/proyecto/venv/lib64/python3.9/site-packages/networkx/utils/backends.py:135: RuntimeWarning: networkx backend defined more than once
: nx_loopback
  backends.update(_get_backends("networkx.backends"))
100% Adding...|██████████| 1/1 [00:30, 30.38s/file]

To track the changes with git, run:
  git add datos.dvc .gitignore

To enable auto staging, run:
  dvc config core.autostage true

(venv) [ec2-user@ip-172-31-14-175 clasificador_edificios]$ qit add datos.dvc .gitignore
(venv) [ec2-user@ip-172-31-14-175 clasificador_edificios]$ dvc push
Collecting
Pushing
  4%|    |Pushing to s3          1.03k/24.8k [00:06<02:05,  190file/s]
  0%|   /home/ec2-user/proyecto/clasificador_edificios/.dvc/cache/files/md5/80/6497141a4bcbed0a9d11c1590e0.00/9.83k [00:00<?, ?B/s]
  0%|   /home/ec2-user/proyecto/clasificador_edificios/.dvc/cache/files/md5/0a/09fc832b967ab11b1blea1788e20.00/7.88k [00:00<?, ?B/s]
  0%|   /home/ec2-user/proyecto/clasificador_edificios/.dvc/cache/files/md5/63/9fa79fcb77bc391fbh2733f9a0.00/25.6k [00:00<?, ?B/s]
  0%|   /home/ec2-user/proyecto/clasificador_edificios/.dvc/cache/files/md5/2c/a5a1de9a46ab823aa0d59599e0.00/28.7k [00:00<?, ?B/s]
  0%|   /home/ec2-user/proyecto/clasificador_edificios/.dvc/cache/files/md5/c5/4ec9348c6466df7f748c90ddff60.00/13.4k [00:00<?, ?B/s]
  0%|   /home/ec2-user/proyecto/clasificador_edificios/.dvc/cache/files/md5/63/a7eb0de3cdf93b6452bdf8faf0.00/8.12k [00:00<?, ?B/s]
  0%|   /home/ec2-user/proyecto/clasificador_edificios/.dvc/cache/files/md5/7e/a2e33df112318a192fce019e900.00/6.78k [00:00<?, ?B/s]
```

Imagen 5. Add de dvc y envío del datastore a S3.

Configuración de dvc a s3

```
ec2-user@ip-172-31-14-175:~/proyecto/clasificador_edificios
(venv) [ec2-user@ip-172-31-14-175 clasificador_edificios]$ ls -lha
total 12K
drwxr-xr-x. 6 ec2-user ec2-user 121 Feb 9 02:59 .
drwxr-xr-x. 4 ec2-user ec2-user 67 Feb 9 02:10 ..
drwxr-xr-x. 3 ec2-user ec2-user 49 Feb 9 03:22 .dvc
-rw-r--r--. 1 ec2-user ec2-user 139 Feb 9 02:59 .gitignore
drwxr-xr-x. 7 ec2-user ec2-user 169 Feb 9 03:17 .git
-rw-r--r--. 1 ec2-user ec2-user 7 Feb 9 02:03 .gitignore
drwxr-xr-x. 2 ec2-user ec2-user 152 Feb 9 02:00 Exploracion_Datos
-rw-r--r--. 1 ec2-user ec2-user 148 Feb 9 01:56 README.md
drwxr-xr-x. 2 ec2-user ec2-user 6 Feb 9 02:04 datos
(venv) [ec2-user@ip-172-31-14-175 clasificador_edificios]$ cat .dvc/config
[core]
remote = aws-remote
['remote "aws-remote"']
url = s3://claseficator-edificios-dvcstore
(venv) [ec2-user@ip-172-31-14-175 clasificador_edificios]$
```

Imagen 6. Configuración de dvc al bucket s3 remoto.

6. Exploración de los datos

El dataset *House Rooms & Streets Image Dataset* contiene un total de 19,658 imágenes en la carpeta *street_data*, en formato JPEG con una resolución uniforme de 224×224 píxeles. El tamaño de los archivos oscila entre 4 KB y 18 KB, con un promedio aproximado de 9 KB, y todas las imágenes se encuentran en modo de color RGB.

De las 19,658 imágenes, se logró clasificar exitosamente **19,354** en ocho categorías: ***apartment, church, garage, house, industrial, office building, retail*** y ***roof***. Las 304 imágenes restantes corresponden a etiquetas minoritarias no contempladas en las categorías principales del dataset (*office, commercial*), por lo que fueron descartadas del análisis.

Como se observa en la siguiente gráfica, el dataset presenta un balance razonable entre clases, con cantidades que varían entre 2,200 y 2,493 imágenes por categoría. La clase *office building* es la que presenta menor representación (2,200), mientras que *apartment* cuenta con la mayor (2,493). Esta diferencia es marginal (~12%), por lo que no se considera un desbalance significativo que requiera técnicas de sobremuestreo o submuestreo.

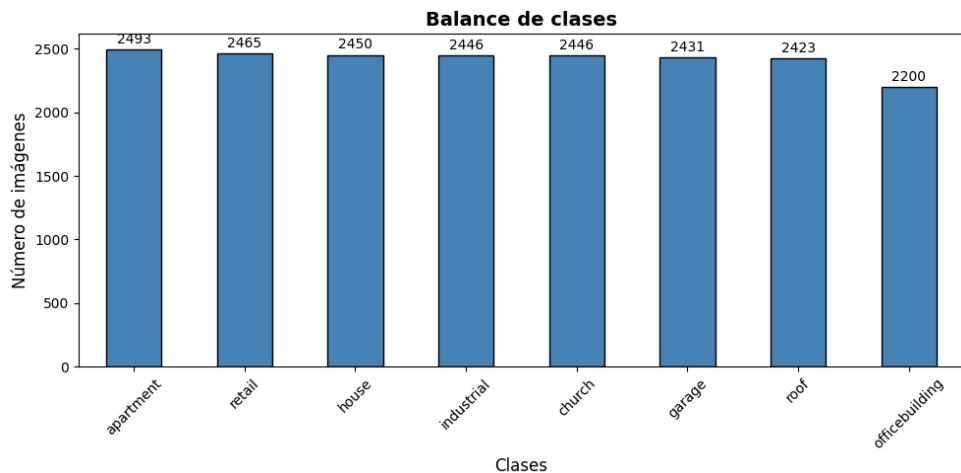


Imagen 7. Balance de clases.

En cuanto a la calidad visual, las imágenes provienen en su mayoría de capturas tipo *Street View*, lo que introduce variabilidad natural en condiciones de iluminación, ángulo de captura y contexto urbano. El análisis del promedio de canales RGB por clase revela perfiles de color similares entre categorías, con valores medios cercanos a 120 en los tres canales, lo que indica **predominancia de tonalidades neutras propias de escenas exteriores**. La clase *office building* presenta los valores RGB más bajos, sugiriendo mayor presencia de sombras o fachadas oscuras en dicha categoría.

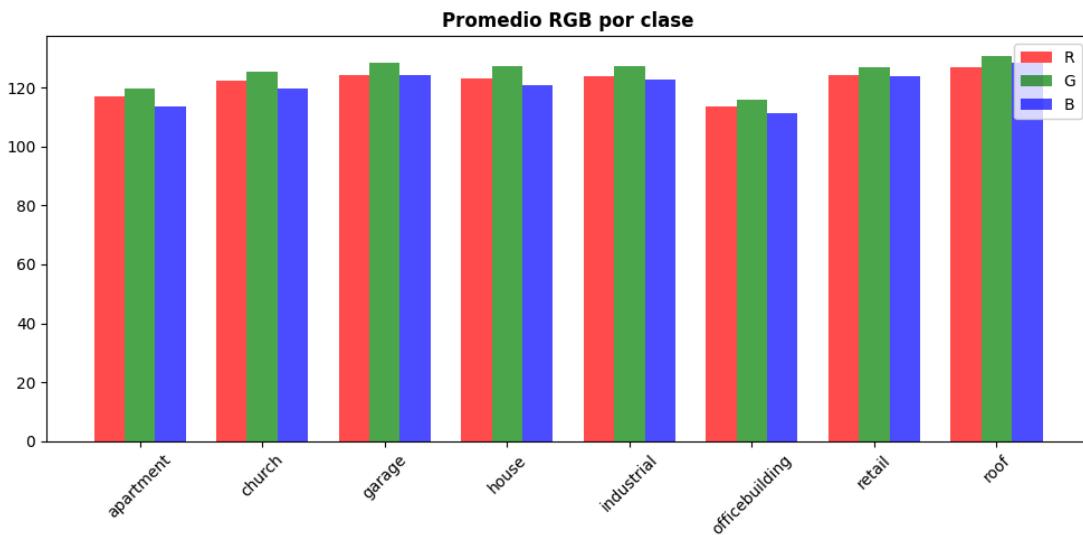


Imagen 8. Promedio RGB por clase.

Se observa que categorías como *house* y *retail* presentan mayor diversidad visual en cuanto a estilos arquitectónicos, colores de fachada y elementos del entorno, mientras que *garage* y *roof* tienden a ser más homogéneas. Adicionalmente, varias imágenes contienen **elementos ajenos al edificio principal (vehículos, vegetación, señalización)**, lo cual puede **representar ruido para el modelo de clasificación**.

A continuación se muestran ejemplos representativos de cada categoría:

Ejemplos de apartment



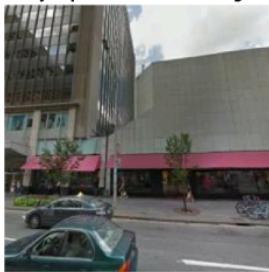
Ejemplos de industrial



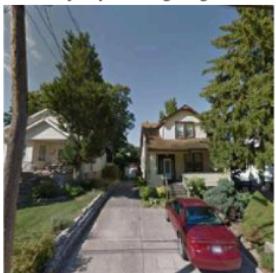
Ejemplos de church



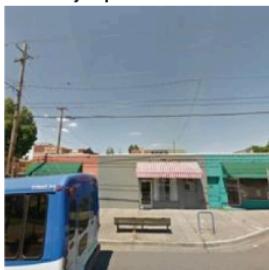
Ejemplos de officebuilding



Ejemplos de garage



Ejemplos de retail



Ejemplos de house



Ejemplos de roof



Imagen 9. Ejemplos dataset.

7. Maqueta del prototipo

El prototipo consiste en un sistema de clasificación de imágenes con los siguientes componentes:

1. **Entrada:** Imagen RGB de una edificación urbana, además un apartado del resultado anteriores.

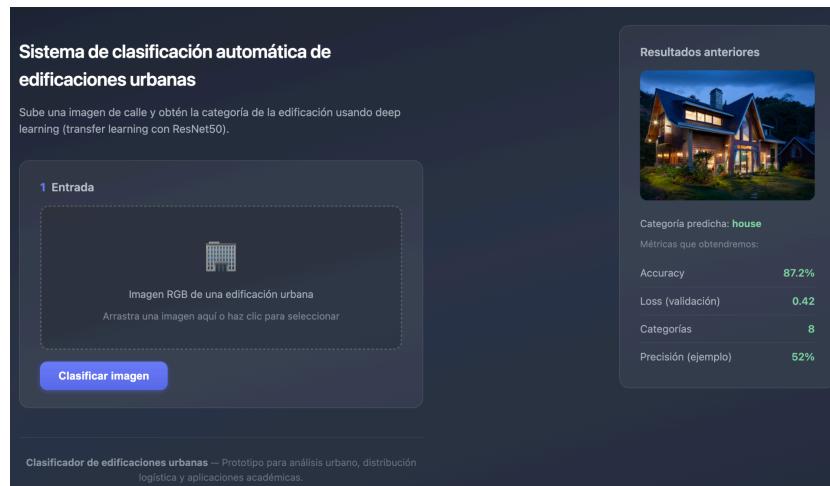


Imagen 10. Prototipo interfaz WEB.

2. **Preprocesamiento y modelo de clasificación:** Se redimensionan las imágenes a 224×224 píxeles y se normalizan los valores de píxel dividiendo por 255 para que queden en el rango $[0, 1]$ antes de pasarlas al modelo.

Se arma un modelo de clasificación con transfer learning: se usa ResNet50 pre entrenada en ImageNet como base (congelada), se le añade una capa de global average pooling y una capa Dense con softmax que devuelve una probabilidad por cada una de las 9 categorías; luego se compila con Adam y entropía cruzada y se entrena una época con los datos de entrenamiento, usando el 20 % de los datos como validación para ver loss y accuracy.

```
import tensorflow as tf

base = tf.keras.applications.ResNet50(weights="imagenet", include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3))
base.trainable = False
model = tf.keras.Sequential([
    base,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(len(CATEGORIES), activation="softmax")
])
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()
```

Imagen 11. Prototipo generación de modelo de clasificación ResNet 50.

Salida: probabilidad por clase y categoría predicha.

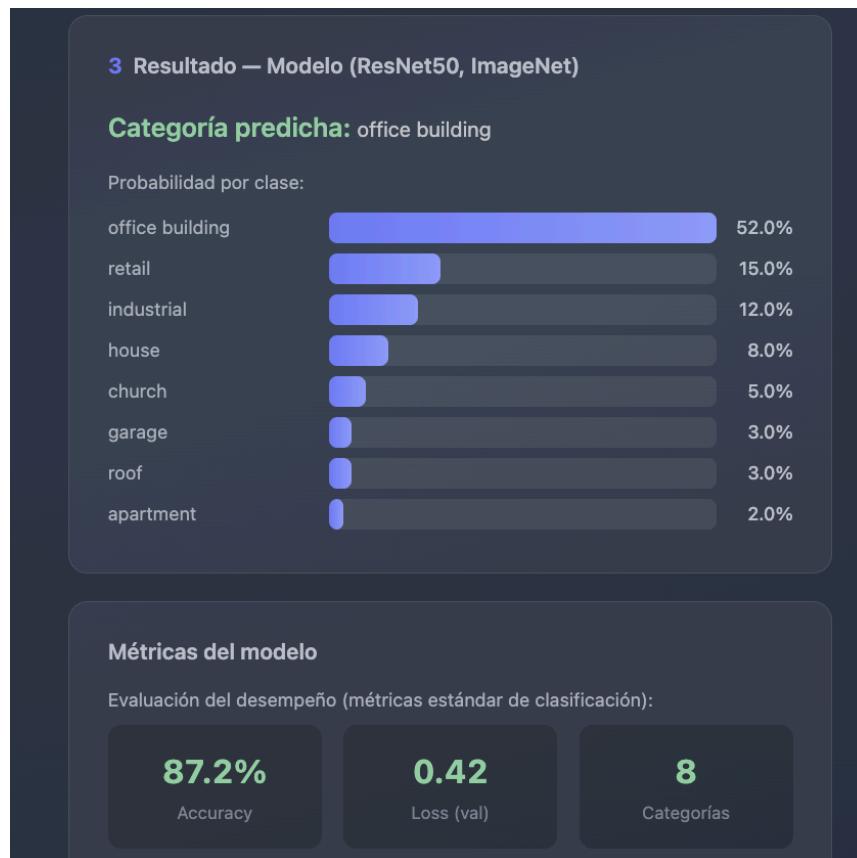


Imagen 12. Prototipo de salida en interfaz WEB.

A nivel conceptual, el prototipo puede integrarse en una interfaz web sencilla o un servicio API para su consumo por aplicaciones externas, aunque el despliegue final no forma parte del alcance actual.

8. Reporte del trabajo en equipo

El desarrollo del proyecto se realizó de manera colaborativa, con distribución clara de responsabilidades:

Actividad	Responsable
Definición del problema y contexto	Todo el equipo
Pregunta de negocio y alcance	Todo el equipo

Análisis y descripción del dataset Maria Paula Acosta Luque

Exploración de los datos (EDA) Geronimo Quiroga

Diseño de la maqueta del prototipo Andrés Torres

Configuración de Git y DVC Marlon Alvarez

Documentación y coordinación Todo el equipo

La comunicación se realizó mediante reuniones periódicas y canales colaborativos, asegurando coherencia técnica y alineación de objetivos durante todo el desarrollo del proyecto.