

Desplegado en la nube de clasificador de Imágenes Climáticas con CNN

Joann Bedoya, Willian Correa, Alejandro Mesa, Jose Restrepo

Diciembre 26, 2024

1 Introducción

Este proyecto tiene como objetivo desplegar un modelo de Machine Learning para la clasificación de fenómenos climáticos mediante una API REST desarrollada con Flask. El modelo, entrenado previamente, utiliza una red convolucional y se encuentra empaquetado en un archivo .h5. Para garantizar la portabilidad, se incluye una configuración de Docker que facilita el despliegue tanto local como en la nube.

1.1 Video youtube

<https://www.youtube.com/watch?v=uvJe98xRPTc>

1.2 Repositorio

<https://github.com/FilosofoCaucano/ImageClasification>

2 Implementación Local

2.1 Backend

2.1.1 Requerimientos:

- **Lenguaje de programación:** Python 3.8.
- **Framework:** Flask para la creación de la API REST.
- **Modelo:** Un modelo convolucional almacenado en `model/climate_classifier.h5`.
- **Dependencias principales:**
 - Flask y flask_cors para la API.
 - TensorFlow para la carga y predicción del modelo.
 - Pillow para procesar imágenes.

2.1.2 Arquitectura del Backend:

El backend está diseñado para:

1. Recibir imágenes a través de solicitudes HTTP POST en el endpoint `/api/image-classification`.
2. Preprocesar las imágenes, incluyendo redimensionamiento y normalización.
3. Realizar inferencias con el modelo convolucional.
4. Responder con la clase climática y el porcentaje de confianza.

2.1.3 Ejecución local:

El backend se ejecuta en el puerto 8080 de la máquina local.

Comando:

```
python app.py
```

2.1.4 Dockerización:

Archivo Dockerfile ejemplo:

```
FROM python:3.10-slim-buster

WORKDIR /todo-app

COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

COPY . .

CMD ["python3", "app.py"]
```

Para construir y ejecutar el contenedor:

```
docker build -t climate-classifier-backend .
docker run -p 8080:8080 climate-classifier-backend
```

2.2 Frontend

2.2.1 Requerimientos:

- **Lenguaje de programación:** JavaScript
- **Framework:** React para la creación del componente del cliente que va a consumir la API
- **Dependencias principales:**
 - React Para el diseño y funcionalidad de la página
 - axios para el consumo de la API.

2.2.2 Arquitectura del Frontend:

El backend está diseñado para:

1. Permitir al usuario cargar una imagen por medio de un input.
2. Enviar imágenes a través de solicitudes HTTP POST en el endpoint `/api/image-classification`.
3. Recibir y mostrar en la página la respuesta recibida por parte del servidor backend

2.2.3 Ejecución local:

El frontend se ejecuta en el puerto 3000 de la máquina local.

Comando:

```
npm start
```

2.2.4 Dockerización:

Archivo Dockerfile ejemplo:

```
# Use the latest LTS version of Node.js
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of your application files
COPY . .

# Expose the port your app runs on
EXPOSE 3000

# Define the command to run your app
CMD ["npm", "start"]
```

Para construir y ejecutar el contenedor:

```
docker build -t climate-classifier-frontend .
docker run -p 3000:3000 climate-classifier-frontend
```

3 Implementación en la Nube: pasos

3.1 Plataforma Seleccionada: Google Cloud Platform (GCP)

3.1.1 Creacion de los proyectos en google cloud:

RECENTES	DESTACADOS	TODOS
		Nombre
✓ ★ ⚡ weatherrecognitionfrontend ?		weatherrecognitionfrontend
★ ⚡ weahterRecognition ?		weahterrecognition

3.1.2 Creacion de repositorios en artifitial registry:

Filtro Ingresar el nombre o el valor de la propiedad						?	☰
<input type="checkbox"/> Nombre ↑	Formato	Tipo	Ubicación	Análisis (Inhabilitado) ⓘ	Descripción	Repo	
<input type="checkbox"/> weatherrecognitionfront2	Docker	Estándar	europe-west4 (Netherlands)	Inhabilitado			

3.1.3 Instalación del cliente de google y login

```
gcloud auth login
```

3.1.4 Creacion de archivos docker front y back

```
WEAHTER_RECOGNITION_API
  model
    climate_classifier.h5
  app.py
Dockerfile
requirements.txt

FROM python:3.10-slim-buster
WORKDIR /todo-app
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
COPY . .
CMD ["python3", "app.py"]

# Use the latest LTS version of Node.js
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
# Install dependencies
RUN npm install
# Copy the rest of your application files
COPY . .
# Expose the port your app runs on
EXPOSE 8080
# Define the command to run your app
CMD ["npm", "start"]
```

3.1.5 Set proyecto

```
gcloud config set project weahterrecognition
```

3.1.6 Subir las imagenes de docker a google cloud

```
gcloud builds submit --tag europe-west4-docker.pkg.dev/
weahterrecognition/weatherrecognition/todoimg:todoimg:tag2
```

```

dbf4e3ba24d5: Preparing
3e01818d79cd: Preparing
dbf4e3ba24d5: Waiting
3e01818d79cd: Waiting
cb0e765bc939: Waiting
241f199fdbb1: Pushed
41ca1d6955b2: Pushed
0e35c2222d0a: Pushed
cb0e765bc939: Pushed
3e01818d79cd: Pushed
dbf4e3ba24d5: Pushed
80e4ed56ffd2: Pushed
937cb58fc0c: Pushed
todotag1: digest: sha256:aa5cf6b6b2940dab788b25d625e89c430db9279bfabb9d43e8fb7148e425d size: 1999
DONE
-----
ID          CREATE_TIME      DURATION SOURCE
IMAGES
STATUS
d27ebd02-6de3-442f-a70c-2d3200c4a72e 2024-12-24T05:58:41+00:00 2M24S gs://weatherrecognitionfrontend_cloudbuild/source/1735019825.9
74635-4d94fa83b827495d851fb58527702fb6.tgz europe-west4-docker.pkg.dev/weatherrecognitionfrontend/weatherrecognitionfront/todotag1:todo
ag1 SUCCESS
3af21159d873: Preparing
c5321f7f53ff: Preparing
df6c1b185b95: Preparing
b23fedba7dbd: Preparing
ae2d55769c5e: Preparing
e2ef8a51359d: Preparing
df6c1b185b95: Waiting
b23fedba7dbd: Waiting
ae2d55769c5e: Waiting
e2ef8a51359d: Waiting
c5321f7f53ff: Layer already exists
df6c1b185b95: Layer already exists
b23fedba7dbd: Layer already exists
ae2d55769c5e: Layer already exists
c1626a4941c8: Pushed
3af21159d873: Pushed
e2ef8a51359d: Layer already exists
7558fd6f6e49: Pushed
bef1eb5146b: Pushed
todotag11: digest: sha256:26b9823e1f182854703958efad89513fb29309ad922153aa99419c12994f0cad size: 2209
DONE
-----
ID          CREATE_TIME      DURATION SOURCE
IMAGES
STATUS
ba601018-9d79-45fe-9d20-1d7ff7d0cc5a 2024-12-24T04:36:21+00:00 2M50S gs://weatherrecognition_cloudbuild/source/1735014970.664638-b3
df94a833794934ba9fdb9d919e9ffd.tgz europe-west4-docker.pkg.dev/weatherrecognition/weatherrecognition/todoimg:todo
tag11: todo tag11 SUCCESS

```

3.1.7 Corremos el proyecto

Corremos la imagen de docker subida a cada uno de los repositorios tanto back como front

3.1.8 Servicios desplegados

Estos son los resultados obtenidos, tenemos que aclarar que el servicio de back subio exitosamente pero el de front tuvo un error a la hora del despliegue.

url back: <https://weatherrecognitionback-463547969270.us-central1.run.app>

<input type="checkbox"/>	<input checked="" type="checkbox"/> weatherrecognitionback	 Contenedor	0	us-central1	autenticacion Permitir sin autenticación
--------------------------	--	--	---	-------------	---

url front: <https://weatherrecognitionfront-428180545004.us-central1.run.app>

<input type="checkbox"/>  Filtro Filtrar servicios					
<input type="checkbox"/>	<input checked="" type="radio"/> Nombre 	Tipo de implementación	Solicitudes/seg 	Región	
<input type="checkbox"/>	 weatherrecognitionfront	 Contenedor	0	us-central1	

4 Pruebas y Validación

4.1 Herramientas Utilizadas:

- **Postman:** Para enviar solicitudes al backend y validar las respuestas.
- **CURL:** Comprobación de conectividad.

4.2 Pruebas locales:

4.2.1 dew

La imagen utilizada fue la siguiente.



Los resultados fueron los siguientes, el modelo predijo la clase con una certeza del 98.05 porcientos.

HTTP <http://localhost:8080/api/image-classification>

POST <http://localhost:8080/api/image-classification>

Params Authorization Headers (10) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> file	File drops.jpg			
Key	Text Value	Description		

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "class": "dew",
3   "confidence": 98.05000305175781
4 }
```

200 OK • 144 ms • 244 B •

4.2.2 sandstorm

La imagen utilizada fue la siguiente.



Los resultados fueron los siguientes, el modelo predijo la clase sandstorm con una certeza del 72.94 poriento.

HTTP <http://localhost:8080/api/image-classification>

POST <http://localhost:8080/api/image-classification>

Params Authorization Headers (6) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> file	File Imagen_prueba5.webp			
Key	Text Value	Description		

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON

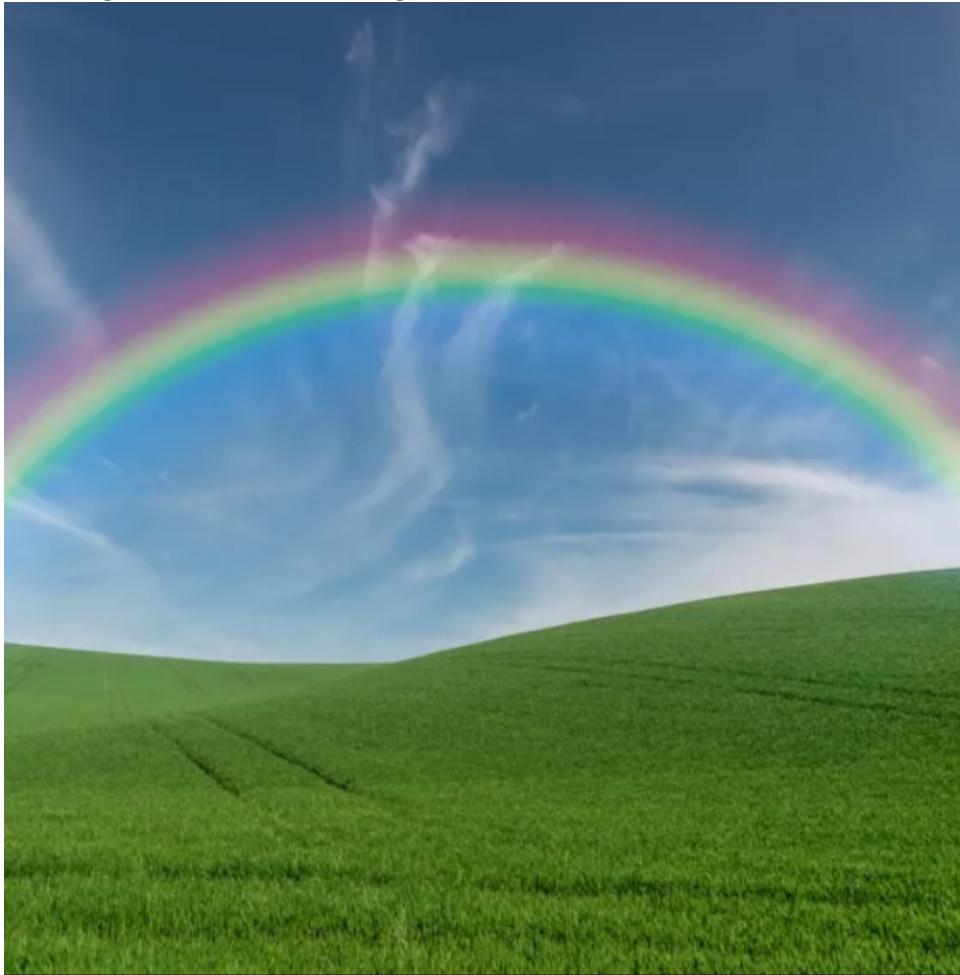
```

1 {
2   "class": "sandstorm",
3   "confidence": 72.94000244149625
4 }
```

200 OK • 453 ms • 250 B •

4.2.3 rainbow

La imagen utilizada fue la siguiente.



Los resultados fueron los siguientes, el modelo predijo la clase rainbow con una certeza del 98.83 porcientos.

http://localhost:8080/api/image-classification

New Request
http://localhost:8080/api/image-classification

Save Share

Send

POST http://localhost:8080/api/image-classification

Params Authorization Headers (10) Body Scripts Settings

None form-data x-www-form-urlencoded raw binary GraphQL

Key	Value	Description	Bulk Edit
file	rainbow.png		
Key	Text	Description	

Body Cookies Headers (6) Test Results

200 OK • 146 ms • 248 B •

Pretty Raw Preview Visualize JSON

```
1 {  
2   "class": "rainbow",  
3   "confidence": 98.83999633789062  
4 }
```

4.3 Prueba frontend

4.4 fogsmog

La imagen utilizada fue la siguiente.



Los resultados fueron los siguientes, el modelo predijo la clase rainbow con una certeza del 55.79 porciento.

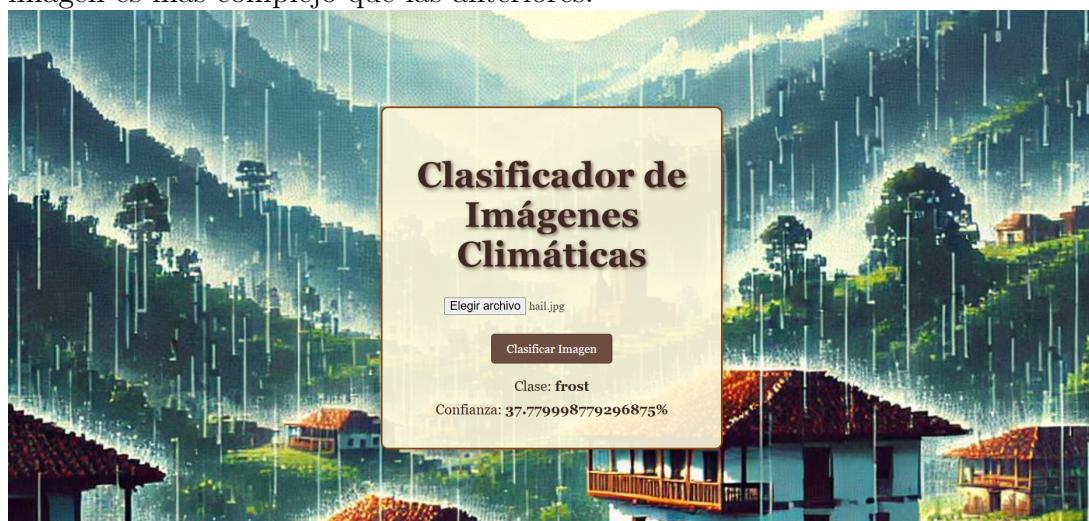


4.5 hail

La imagen utilizada fue la siguiente.



Los resultados fueron los siguientes, el modelo predijo la clase frost con una certeza del 37.77 porciento, no era el resultado que esperábamos aunque el reconocimiento de esta imagen es mas complejo que las anteriores.



5 Conclusiones

- El modelo es bastante preciso y acierta con más de un 60% en la mayoría de las peticiones.
- En las imágenes que contienen colores blancos, como frost, hail, rime, es donde más se nota la dificultad del modelo para categorizar estas imágenes.