

Documentación del Sistema de Detección de Maquinaria en Obras de Construcción para la CGR

Bienvenid@ a la documentación del sistema de detección de maquinaria en obras de construcción para la Contraloría General de la República. En esta documentación encontrarás información detallada sobre la arquitectura del modelo de detección de objetos, la arquitectura del sistema dentro de AWS y la arquitectura de la imagen de Docker.

Índice

- [Arquitectura del Modelo](#)
- [Proceso de Entrenamiento](#)
- [Configuración del Sistema en AWS](#)
- [Configuración de Google Drive API](#)
- [Estructura del Docker](#)

Arquitectura del Modelo

El modelo de detección de objetos en imágenes de maquinaria en obras de construcción se basa en FasterRCNN con ResNet50 como red base. FasterRCNN es un modelo de detección de objetos que consta de dos subredes: una red base y una red de detección. La red base es una red convolucional que se encarga de extraer características de la imagen de entrada, mientras que la red de detección es una red que se encarga de realizar la detección de objetos en la imagen.

ResNet50 es una red convolucional profunda que consta de 50 capas. Esta red es utilizada como red base en el modelo, ya que ha demostrado ser efectiva en la extracción de características de imágenes. FasterRCNN con ResNet50 como red base es un modelo de detección de objetos que ha demostrado ser efectivo en la detección de objetos en imágenes de maquinaria en obras de construcción.

La razón principal por la que se eligió ResNet50 como red base es porque es una red profunda que ha sido pre-entrenada en un gran conjunto de datos de imágenes. Esto significa que la red base ya ha aprendido a extraer características de alto nivel de las imágenes, lo que permite detectar objetos en las imágenes con alta precisión. Además, ResNet50 es una red profunda que ha demostrado ser efectiva en la extracción de características de imágenes, lo que la hace ideal para su uso como red base en el modelo de detección de objetos. Comparado con otros modelos, ResNet50 es más rápido y preciso en la detección de objetos en imágenes de maquinaria en obras de construcción.

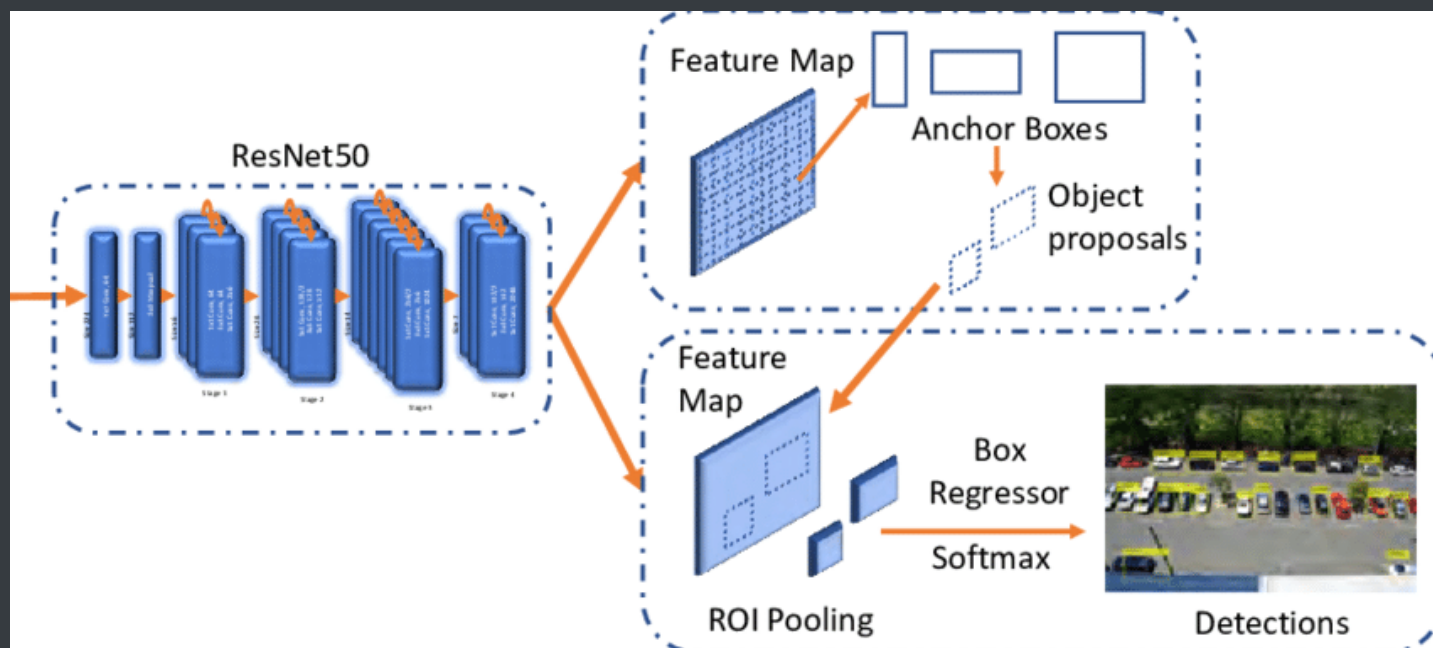
El modelo de detección de objetos consta de las siguientes etapas:

1. **Extracción de características:** La red base (ResNet50) se encarga de extraer características de la imagen de entrada. La red base es una red convolucional profunda que ha sido pre-entrenada en un gran conjunto de datos de imágenes. Esta red es capaz de extraer características de alto nivel de la imagen de entrada, lo que permite detectar

objetos en la imagen.

2. **Propuestas de regiones:** Se generan propuestas de regiones en la imagen que pueden contener objetos. Estas propuestas se generan utilizando una red de detección que se entrena para predecir las regiones en las que es más probable que se encuentren objetos.
3. **Extracción de características de las regiones propuestas:** Se extraen características de las regiones propuestas utilizando la red base. Estas características se utilizan para clasificar las regiones propuestas en las diferentes clases de objetos.
4. **Clasificación de las regiones propuestas:** Se clasifican las regiones propuestas en las diferentes clases de objetos. Para ello, se utiliza un clasificador que se entrena para predecir la clase de objeto presente en cada región propuesta.
5. **Predicción de las cajas delimitadoras de los objetos:** Se predicen las cajas delimitadoras de los objetos en las regiones propuestas. Estas cajas delimitadoras se utilizan para localizar los objetos en la imagen y dibujar las cajas delimitadoras alrededor de los objetos detectados.

La ejemplificación de la arquitectura del modelo se muestra en la siguiente figura:



Una arquitectura como esta permite detectar y clasificar objetos en imágenes con alta precisión y rapidez, dando resultados similares a los que se observan en la siguiente imagen:



En nuestro caso, el modelo fue entrenado con imágenes que poseían uno o varios objetos representados pertenecientes a las siguientes clases o *labels*:

- `dump_truck`
- `person`
- `excavator`
- `loader`
- `mixer_truck`
- `steamroller`

[Volver al inicio](#)

Proceso de Entrenamiento

Para el proceso de entrenamiento de un modelo de detección de objetos basado en Faster R-CNN con ResNet-50, se utilizó PyTorch y torchvision. A continuación, se detalla la arquitectura y el flujo de trabajo para el entrenamiento, evaluación y visualización de los resultados.

Documentación del Código de Entrenamiento

Este código implementa un modelo de detección de objetos utilizando PyTorch y el modelo pre-entrenado Faster R-CNN basado en ResNet-50.

```
import torch
import torchvision
from torchvision.models.detection import fasterrcnn_resnet50_fpn,
FasterRCNN_ResNet50_FPN_Weights
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.transforms import functional as F
from torch.utils.data import Dataset, DataLoader
from torch.cuda.amp import GradScaler, autocast
from PIL import Image
import os
import json
import torchvision.transforms as transforms

os.environ['PYTORCH_CUDA_ALLOC_CONF'] = 'max_split_size_mb:128'
os.environ['PYTORCH_CUDA_ALLOC_CONF'] = 'expandable_segments:True'

annotations_file = 'path/Train_dataset.json'

def class_text_to_int(class_text):
```

```

class_mapping = { "dump_truck": 1, "person": 2, "excavator":
3, "loader": 4, "mixer_truck": 5, "steamroller": 6}
return class_mapping.get(class_text, None)

class CustomDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None):
        with open(annotations_file) as f:
            self.img_labels = json.load(f)
        self.img_dir = img_dir
        self.transform = transform if transform is not None else
transforms.Compose([
            transforms.ToTensor(), # Converts PIL images to PyTorch
tensors
        ])

    def __len__(self):
        return len(self.img_labels['images'])

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels['images']
[idx]['image_path'])
        image = Image.open(img_path).convert("RGB")
        # Apply transformations
        if self.transform:
            image = self.transform(image)
        objects = self.img_labels['images'][idx]['objects']
        boxes = []
        labels = []
        for obj in objects:
            xmin, ymin, xmax, ymax = obj['bbox'].values()
            boxes.append([xmin, ymin, xmax, ymax])
            labels.append(class_text_to_int(obj['class']))
        boxes = torch.as_tensor(boxes, dtype=torch.float32)

```

```

        labels = torch.as_tensor(labels, dtype=torch.int64)
        target = {"boxes": boxes, "labels": labels}
        return image, target

def get_model(num_classes):
    model =
fasterrcnn_resnet50_fpn(weights=FasterRCNN_ResNet50_FPN_Weights.DEFAULT)
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features,
num_classes)
    return model

def collate_fn(batch):
    return tuple(zip(*batch))

def train_one_epoch(model, optimizer, data_loader, device, epoch,
accumulation_steps=4):
    model.train()
    scaler = GradScaler()
    optimizer.zero_grad()
    for i, (images, targets) in enumerate(data_loader):
        images = [image.to(device) for image in images]
        targets = [{k: v.to(device) for k, v in t.items()} for t in
targets]
        with autocast():
            loss_dict = model(images, targets)
            losses = sum(loss for loss in loss_dict.values()) /
accumulation_steps
            scaler.scale(losses).backward()
            if (i + 1) % accumulation_steps == 0 or (i + 1) ==
len(data_loader):

```

```

        scaler.step(optimizer)
        scaler.update()
        optimizer.zero_grad()
    print(f"Epoch #{epoch} pérdida: {losses.item()}")

def main():
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    num_classes = 7
    dataset = CustomDataset(annotations_file, 'path/Images_etiquetadas',
transform=None)

    data_loader = DataLoader(dataset, batch_size=24, shuffle=True,
collate_fn=collate_fn, num_workers=20, pin_memory=True)

    model = get_model(num_classes).to(device)
    optimizer = torch.optim.SGD(model.parameters(), lr=0.005,
momentum=0.9)
    #optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

    num_epochs = 100
    for epoch in range(num_epochs):
        train_one_epoch(model, optimizer, data_loader, device, epoch)
        torch.save(model.state_dict(), f'path/model_epoch_{epoch}.pth',)

if __name__ == "__main__":
    main()

```

1. Importaciones:

Importa las librerías necesarias incluyendo PyTorch, torchvision, PIL para manejo de imágenes, y otras utilidades de sistema y JSON para manejar archivos y configuraciones.

2. Configuraciones de CUDA:

Establece variables de entorno para optimizar la asignación de memoria en CUDA de PyTorch.

3. Archivo de anotaciones:

Define la ruta al archivo JSON que contiene las anotaciones del dataset.

4. Clase `CustomDataset` :

- **Constructor:** Carga las anotaciones de un archivo JSON y establece el directorio de imágenes y transformaciones opcionales.
- `__len__` : Retorna el número de imágenes en el dataset.
- `__getitem__` : Carga y transforma una imagen del dataset, extrae las cajas de delimitación y las etiquetas de las clases de los objetos presentes en la imagen, y retorna la imagen y los objetivos (etiquetas y cajas).

5. Funciones:

- **Función `class_text_to_int` :**
Convierte el texto de la clase de un objeto a un entero según un mapeo predefinido.
- **Función `get_model` :**
Carga el modelo Faster R-CNN preentrenado y modifica el predictor para ajustarse al número de clases proporcionado.
- **Función `collate_fn` :**
Función auxiliar para agrupar los datos procesados en un batch.
- **Función `train_one_epoch` :**
Entrena el modelo por una época. Utiliza GradScaler para manejar la precisión mixta durante el entrenamiento, ayudando a mejorar el rendimiento y reducir el uso de memoria. Realiza la propagación hacia atrás y actualiza los pesos del modelo.
- **Función `main` :**
Establece el dispositivo de entrenamiento (GPU o CPU). Prepara el dataset y el cargador de datos. Crea el modelo y el optimizador. Ejecuta el entrenamiento del modelo para un número definido de épocas y guarda los pesos del modelo después de cada época.

6. **Bloque de ejecución:** Verifica si el script se ejecuta como archivo principal y, de ser así, ejecuta la función main.

Documentación de Código de Evaluación

Este código implementa un proceso de evaluación para el modelo de objetos basado en Faster R-CNN con ResNet-50 usando PyTorch y torchvision.

```
import torch
import torchvision
from torchvision.models.detection import fasterrcnn_resnet50_fpn,
FasterRCNN_ResNet50_FPN_Weights
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.transforms import functional as F
from torch.utils.data import Dataset, DataLoader
from PIL import Image
import os
import json
import torchvision.transforms as transforms
import numpy as np

def collate_fn(batch):
    return tuple(zip(*batch))

def class_text_to_int(class_text):
    class_mapping = {
        "dump_truck": 1,
        "person": 2,
        "excavator": 3,
        "loader": 4,
        "mixer_truck": 5,
        "steamroller": 6
    }
    return class_mapping.get(class_text, None)

class CustomDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None):
```

```

        with open(annotations_file) as f:
            self.img_labels = json.load(f)
        self.img_dir = img_dir
        self.transform = transform if transform is not None else
transforms.Compose([
            transforms.ToTensor(),
        ])

    def __len__(self):
        return len(self.img_labels['images'])

    def __getitem__(self, idx):
        img_info = self.img_labels['images'][idx]
        img_path = os.path.join(self.img_dir, img_info['image_path'])
        image = Image.open(img_path).convert("RGB")

        if self.transform:
            image = self.transform(image)

        boxes = []
        labels = []
        for obj in img_info['objects']:
            bbox = obj['bbox']
            boxes.append([bbox['x_min'], bbox['y_min'], bbox['x_max'],
bbox['y_max']])
            labels.append(class_text_to_int(obj['class']))

        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        labels = torch.as_tensor(labels, dtype=torch.int64)
        target = {"boxes": boxes, "labels": labels}

        return image, target

```

```

def bbox_iou(box1, box2):
    x1 = max(box1[0], box2[0])
    y1 = max(box1[1], box2[1])
    x2 = min(box1[2], box2[2])
    y2 = min(box1[3], box2[3])
    inter_area = max(x2 - x1, 0) * max(y2 - y1, 0)
    box1_area = (box1[2] - box1[0]) * (box1[3] - box1[1])
    box2_area = (box2[2] - box2[0]) * (box2[3] - box2[1])
    iou = inter_area / float(box1_area + box2_area - inter_area)
    return iou

def get_model(num_classes):
    model =
    fasterrcnn_resnet50_fpn(weights=FasterRCNN_ResNet50_FPN_Weights.DEFAULT)
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features,
num_classes)
    return model

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

Pocosi_24042024_30epocs = 'path/model.pth'
model = get_model(num_classes=7)
model.load_state_dict(torch.load(Pocosi_24042024_30epocs,
map_location=device))
model = model.to(device)
model.eval()

test_annotations_file = 'path/test_dataset.json'
images_path = 'path/Images_etiquetadas'
test_dataset = CustomDataset(test_annotations_file, images_path)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn)

```

```

ious = []
true_positives, false_positives, false_negatives = 0, 0, 0
iou_threshold = 0.7

for images, targets in test_loader:
    images = [image.to(device) for image in images]
    outputs = model(images)
    for i, output in enumerate(outputs):
        predicted_boxes = output['boxes'].data.cpu().numpy()
        predicted_labels = output['labels'].data.cpu().numpy()
        target_boxes = targets[i]['boxes'].data.cpu().numpy()
        target_labels = targets[i]['labels'].data.cpu().numpy()

        matched_gt = set()
        for pb_idx, predicted_box in enumerate(predicted_boxes):
            for tb_idx, target_box in enumerate(target_boxes):
                iou = bbox_iou(predicted_box, target_box)
                if iou > iou_threshold and tb_idx not in matched_gt:
                    matched_gt.add(tb_idx)
                    ious.append(iou)
                    if predicted_labels[pb_idx] == target_labels[tb_idx]:
                        true_positives += 1
                    else:
                        false_positives += 1
                    break
            else:
                false_positives += 1
        false_negatives += len(target_boxes) - len(matched_gt)

average_iou = sum(ious) / len(ious) if ious else 0
precision = true_positives / (true_positives + false_positives) if
true_positives + false_positives > 0 else 0
recall = true_positives / (true_positives + false_negatives) if
true_positives + false_negatives > 0 else 0

```

```
print(f"Average IoU: {average_iou:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

1. **Importaciones:** Se importan las librerías necesarias como PyTorch, torchvision, PIL para manejo de imágenes, numpy para operaciones numéricas, y otras para el manejo de archivos y transformaciones
2. **Funciones y Clases:**
 - **collate_fn** : Función que ayuda a agrupar los elementos de datos en un lote, necesario para el cargador de datos.
 - **class_text_to_int** : Mapea el texto de clasificación de objetos a un número entero para su procesamiento.
 - **CustomDataset** : Define un dataset personalizado que carga y transforma imágenes y etiquetas de un archivo JSON, y prepara los datos para el entrenamiento o evaluación.
 - **bbox_iou** : Calcula la Intersección sobre Unión (IoU) de dos cajas delimitadoras, útil para evaluar la precisión de las predicciones del modelo.
 - **get_model** : Carga y modifica un modelo Faster R-CNN preentrenado para adaptarlo al número específico de clases.
3. **Carga del Modelo:**
 - Establece el dispositivo de cálculo (CPU o GPU).
 - Carga los pesos de un modelo previamente entrenado desde un archivo guardado.
 - Cambia el modelo al modo de evaluación, desactivando características como Dropout o BatchNorm durante las predicciones.
4. **Definición del Dataset de Prueba:**
 - Define el archivo de anotaciones y la ruta de las imágenes para el conjunto de datos de prueba.
 - Carga el dataset de prueba usando la clase `CustomDataset` y prepara un cargador de datos para iterar sobre el dataset.

5. Evaluación del Modelo:

- Inicializa variables para contar verdaderos positivos, falsos positivos, y falsos negativos, y calcula la precisión (precision), el recall y el IoU medio de las predicciones.
- Itera sobre el conjunto de datos de prueba, realiza predicciones con el modelo y compara las cajas predichas con las cajas verdaderas basándose en el umbral de IoU.
- Calcula y actualiza las métricas de verdaderos positivos, falsos positivos y falsos negativos basándose en la coincidencia de las etiquetas y la IoU.
- Al final del proceso, imprime el IoU medio, la precisión y el recall.

Código para Visualizar los Resultados

Este código en Python utiliza Matplotlib para visualizar predicciones y verdades de tierra (ground truth) de un modelo de detección de objetos sobre imágenes. A continuación, la explicación en español del código:

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Función para graficar las imágenes
def draw_boxes(ax, boxes, labels, color):
    for box, label in zip(boxes, labels):
        x1, y1, x2, y2 = box
        rect = patches.Rectangle((x1, y1), x2 - x1, y2 - y1, linewidth=2,
                                edgecolor=color, facecolor='none')
        ax.add_patch(rect)
        ax.text(x1, y1, label, verticalalignment='bottom',
                horizontalalignment='left', color=color, fontsize=12)

# Figura de visualización
fig, axes = plt.subplots(20, 1, figsize=(30, 50)) # 1 row, 5 columns
image_counter = 0
```

```
test_annotations_file = 'path/test_dataset.json'
```

```
# Carga datos
```

```
test_dataset = CustomDataset(test_annotations_file, images_path,  
transform=None)
```

```
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False,  
collate_fn=collate_fn)
```

```
for images, targets in test_loader:
```

```
    images = [image.to(device) for image in images]
```

```
    outputs = model(images)
```

```
    for i, image in enumerate(images):
```

```
        if image_counter >= 20:
```

```
            break
```

```
        ax = axes[image_counter]
```

```
        img = F.to_pil_image(image.cpu())
```

```
        ax.imshow(img)
```

```
# Cajas reales
```

```
target_boxes = targets[i]['boxes'].cpu().numpy()
```

```
target_labels = [str(label) for label in targets[i]
```

```
['labels']].cpu().numpy()]
```

```
draw_boxes(ax, target_boxes, target_labels, 'blue')
```

```
#Cajas Predichas
```

```
predicted_boxes = outputs[i]['boxes'].detach().cpu().numpy()
```

```
predicted_scores = outputs[i]['scores'].detach().cpu().numpy()
```

```
predicted_labels = [str(label) for label in outputs[i]
```

```
['labels']].detach().cpu().numpy()]
```



```

        high_score_idx = [idx for idx, score in
enumerate(predicted_scores) if score > 0.7]
        predicted_boxes = predicted_boxes[high_score_idx]
        predicted_labels = [predicted_labels[idx] for idx in
high_score_idx]

        draw_boxes(ax, predicted_boxes, predicted_labels, 'red')

        ax.axis('off')
        image_counter += 1

    if image_counter >= 20:
        break

plt.tight_layout()
plt.show()

```

1. **Importaciones:** Se importa `matplotlib.pyplot` para la creación de gráficos y `matplotlib.patches` para dibujar rectángulos, que representarán las cajas delimitadoras.
2. **Función `draw_boxes` :** Dibuja cajas delimitadoras sobre una imagen y etiqueta cada caja. Recibe un eje de Matplotlib (`ax`), las cajas (`boxes`), las etiquetas asociadas (`labels`) y un color especificado (`color`). Por cada caja y etiqueta:
 - Calcula las coordenadas del rectángulo.
 - Crea y añade un rectángulo al eje.
 - Añade texto que representa la etiqueta de la clase en la esquina superior izquierda de la caja.
3. **Inicialización de la Figura:**
 - Se crea una figura y un conjunto de ejes usando `subplots` , configurados para mostrar 20 imágenes en una columna.

4. Carga de Datasets y DataLoader:

- Define el archivo y ruta para el conjunto de datos de prueba.
- Utiliza la clase `CustomDataset` para cargar el conjunto de datos y el `DataLoader` para iterar sobre él.

5. Iteración sobre el DataLoader:

- Para cada lote de imágenes y objetivos del DataLoader:
 - Se transfieren las imágenes al dispositivo adecuado (GPU o CPU).
 - Se obtienen las predicciones del modelo.
 - Para cada imagen y sus predicciones:
 - Si ya se han procesado 20 imágenes, se detiene el bucle.
 - Se convierte la imagen tensorial a imagen PIL para visualización.
 - Se visualizan las cajas delimitadoras de las verdades de tierra (color azul) y las predicciones (color rojo), utilizando la función `draw_boxes`.
 - Se filtran las predicciones para mostrar solo aquellas con un puntaje superior al 70%.
 - Se desactivan los ejes para una visualización más limpia.

6. Visualización de Resultados:

- Se ajustan automáticamente los subplots para que se adapten al layout de la figura.
- Se muestra la figura con todas las imágenes y sus correspondientes cajas delimitadoras superpuestas.

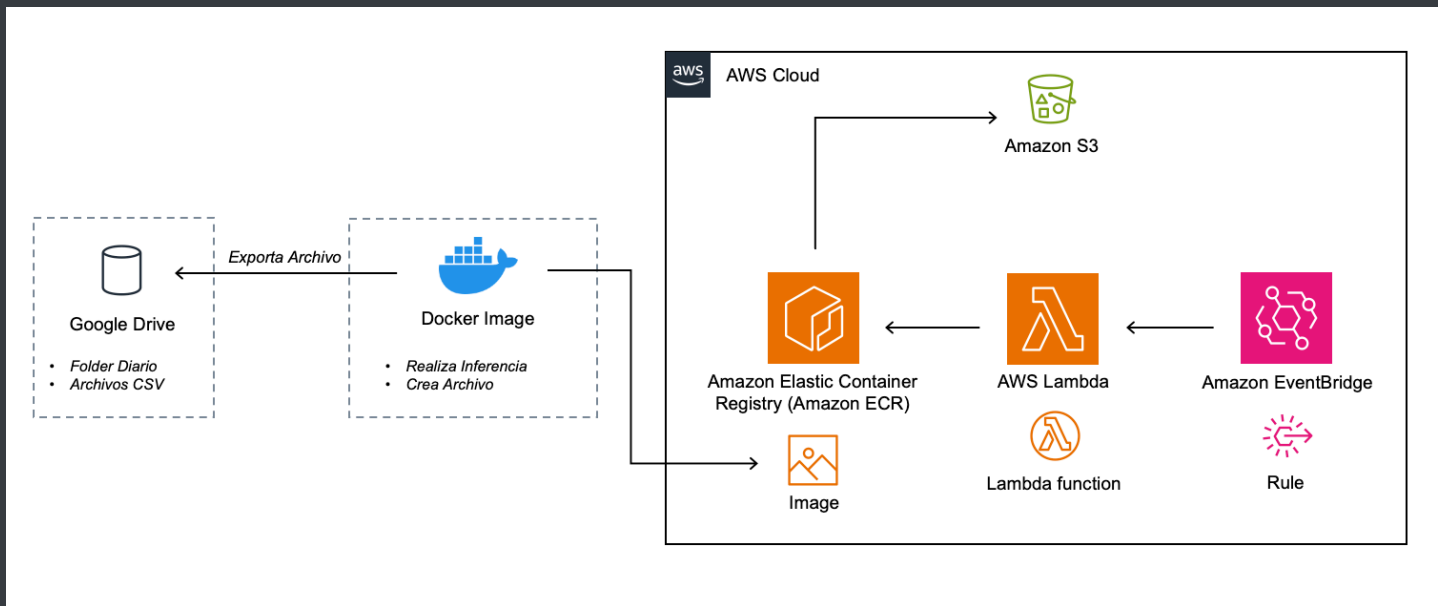
[Volver al inicio](#)

Configuración del Sistema en AWS

El sistema de detección de maquinaria en obras de construcción para la Contraloría General de la República se encuentra desplegado en la nube de Amazon Web Services (AWS). La arquitectura del sistema en AWS se compone de los siguientes servicios:

1. **Amazon Elastic Container Registry (ECR):** ECR es un servicio de AWS que permite almacenar, administrar y desplegar imágenes de contenedores de Docker. En el sistema, se utiliza ECR para almacenar la imagen de Docker que contiene el modelo de detección de objetos y los archivos de configuración necesarios para realizar las inferencias.
2. **Amazon Lambda:** Lambda es un servicio de AWS que permite ejecutar código sin tener que aprovisionar ni administrar servidores. En el sistema, se utiliza Lambda para ejecutar el modelo de detección de objetos y realizar las inferencias sobre las imágenes de maquinaria en obras de construcción.
3. **Amazon EventBridge:** EventBridge es un servicio de AWS que permite conectar aplicaciones con eventos generados por servicios de AWS, aplicaciones de software o eventos personalizados. En el sistema, se utiliza EventBridge para programar la ejecución del modelo de detección de objetos a través de un *scheduler*.
4. **Amazon IAM:** IAM es un servicio de AWS que permite gestionar el acceso a los recursos de AWS de forma segura. En el sistema, se utiliza IAM para definir los roles y permisos necesarios para que los servicios de AWS puedan interactuar entre sí.
5. **Amazon S3:** S3 es un servicio de AWS que permite almacenar y recuperar datos de forma segura y escalable. En el sistema, se utiliza S3 para almacenar el token.pickle que permite acceder a la API de Google Drive y las imágenes de maquinaria en obras de construcción sobre las cuales se realizarán las inferencias.

La arquitectura de AWS se muestra representada visualmente en la siguiente figura:



Configuración de los Servicios en AWS

A continuación se detallan los pasos necesarios para configurar el cada uno de los servicios en AWS:

Amazon Elastic Container Registry (ECR)

Instalación de AWS Tools para PowerShell

1. Abrir PowerShell con derechos de administrador:

- Para instalar AWS Tools para PowerShell, ejecute el siguiente comando:

```
Install-Module -Name AWS.Tools.Common  
Install-Module -Name AWS.Tools.ECR
```

- Si se necesita configurar una versión anterior, utilizar:

```
Install-Module -Name AWSPowerShell
```

2. Importar el Módulo AWS PowerShell:

- Luego de la instalación, importar el módulo con:

```
Import-Module AWS.Tools.Common
Import-Module AWS.Tools.ECR
```

3. Verificación de la Instalación:

- Comprobar que el cmdlet `Get-ECRLoginCommand` está disponible mediante:

```
Get-Command Get-ECRLoginCommand
```

- En caso de tener errores al ejecutar los comandos de importación tras la instalación, es posible que sea a causa de los permisos de Windows. Para solucionar esto, se deben ejecutar los siguientes comandos:

```
Get-ExecutionPolicy
Set-ExecutionPolicy RemoteSigned
Import-Module AWS.Tools.Common
Import-Module AWS.Tools.ECR
Get-Command Get-ECRLoginCommand
```

- El resultado debería de ser similar a la imagen:

```
PS C:\Windows\system32> Get-Command Get-ECRLoginCommand

CommandType      Name                               Version      Source
-----
Cmdlet           Get-ECRLoginCommand              4.1.538     AWS.Tools.ECR

PS C:\Windows\system32>
```

4. Configuración de Credenciales de AWS:

- Una vez el usuario IAM esté creado, obtener las claves de acceso. Configurar las credenciales con:

```
aws configure
```

- Establecer las credenciales en PowerShell, reemplazando `YOUR_ACCESS-KEY` y `YOUR_SECRET-KEY` por las credenciales correspondientes:

```
Set-AWSCredential -AccessKey YOUR_ACCESS_KEY -SecretKey  
YOUR_SECRET_KEY -StoreAs default
```

5. Inicio de Sesión en ECR:

- Utilizar el comando para iniciar sesión en ECR con Docker:

```
(Get-ECRLoginCommand -Region us-east-1).Password | docker login --  
username AWS --password-stdin YOUR_ID.dkr.ecr.us-east-  
1.amazonaws.com
```

- El resultado debería de ser similar a la imagen:

```
PS C:\Windows\system32> (Get-ECRLoginCommand -Region us-east-1).Password | docker login --username AWS --password-stdin YOUR_ID.dkr.ecr.us-east-1.amazonaws.com  
Login Succeeded  
PS C:\Windows\system32>
```

Uso de Docker con ECR

Finalmente, se puede navegar al path donde se encuentra el archivo `Dockerfile` y ejecutar los comandos proveídos por ECR, como el comando para construir la imagen de Docker:

```
docker build -t contraloria_testecr .
```

El resultado debería de ser similar a la imagen:

```

PS > docker build -t contraloria_testecr .
2024/03/17 13:41:11 http2: server: error reading preface from client //./pipe/docker_engine: file has already been closed
[+] Building 1.5s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 203B
=> [internal] load metadata for docker.io/pytorch/pytorch:2.2.1-cuda12.1-cudnn8-runtime
=> [auth] pytorch/pytorch:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 52B
=> [1/4] FROM docker.io/pytorch/pytorch:2.2.1-cuda12.1-cudnn8-runtime@sha256:11691e035a3651d25a87116b4f6adc113a27a29d8f5a6a583f8569e0ee5ff897
=> [internal] load build context
=> => transferring context: 275B
=> CACHED [2/4] WORKDIR /inference
=> CACHED [3/4] COPY . /inference
=> CACHED [4/4] RUN pip install -r requirements.txt
=> exporting to image
=> => exporting layers
=> => writing image sha256:8e949289cfdc131ddebd55c10d16f3090a791a29fe9a8978a18cf13aaf1ef6a6
=> => naming to docker.io/library/contraloria_testecr

View build details: docker-desktop://dashboard/build/default/default/qigbtvdvd6iej15epe9o0fwb15

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
PS > docker tag contraloria_testecr:latest [redacted].dkr.ecr.us-east-1.amazonaws.com/contraloria_testecr:latest
PS > docker push [redacted].dkr.ecr.us-east-1.amazonaws.com/contraloria_testecr:latest
The push refers to repository [redacted].dkr.ecr.us-east-1.amazonaws.com/contraloria_testecr]
2767d755c963: Pushing [=====>] 159.8MB/177.7MB
98a491be1225: Pushing [=====>] 76.86MB/165.8MB
45a4f95ce271: Pushed
ca12c2d49851: Pushed
5f70bf18a086: Pushed
bfb59c8a0e28: Pushing [>] 46.12MB/7.496GB
472907e9bcfb: Pushed
d101c9453715: Pushed

```

Amazon Lambda

Utilización de la Imagen Docker desde Lambda

Una vez que la imagen de Docker ha sido construida y subida a ECR, se puede utilizar desde Lambda. Para ello, se debe crear una función de Lambda y configurarla para que utilice la imagen de Docker almacenada en ECR, con las siguientes configuraciones:

[Lambda](#) > [Functions](#) > Create function

Create function [Info](#)

Choose one of the following options to create your function.

☐ Author from scratch
Start with a simple Hello World example.

☐ Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

☒ Container image
Select a container image to deploy for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Container image URI [Info](#)
The location of the container image to use for your function.

Requires a valid Amazon ECR image URI.


Browse images

Amazon EventBridge

Crear Reglas de Evento

Se configura una regla de evento en EventBridge para programar la ejecución de la función de Lambda que utiliza la imagen de Docker almacenada en ECR. Esta regla se configura para que se ejecute periódicamente según un cronograma definido, por ejemplo, todos los días a las 8 PM UTC, que corresponde a las 6 PM en Costa Rica, tal y como se ve en la imagen:

☐ Trigger

**EventBridge (CloudWatch Events):** [6pmrule](#)

Rule state: **ENABLED**

▼ Details

☐ Description: **This is to trigger the lambda at 6pm**
Event bus: **default**
isComplexStatement: **No**
name: **6pmrule**
Schedule expression: **cron(0 20 * * ? *)**
Service principal: **events.amazonaws.com**
Statement ID: **[REDACTED]**
url: **events/home#/rules/6pmrule**

Amazon IAM

Crear y Gestión de Políticas

Se utiliza IAM para definir políticas de seguridad que controlen el acceso a los recursos de AWS. Esto incluye la creación de roles y la asignación de políticas a esos roles para permitir o restringir acciones específicas en los servicios de AWS, tal y como se ve en la imagen:

Identity and Access Management (IAM)

Search IAM

Dashboard

Access management

User groups

Users

Roles

Policies

Identity providers

Account settings

Access reports

Access Analyzer

External access

Unused access

Analyzer settings

Policy was successfully attached to role.

Creation date

March 17, 2024, 20:14 (UTC-06:00)

Last activity

18 hours ago

ARN

Maximum session duration

1 hour

Permissions

Trust relationships

Tags

Access Advisor

Revoke sessions

Permissions policies (2) Info

Simulate

Remove

Add permissions



You can attach up to 10 managed policies.

Search

Filter by Type

All types

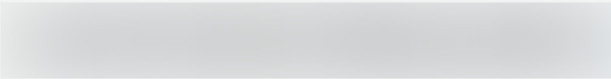
< 1 >

<input type="checkbox"/>	Policy name	Type	Attached entities
<input type="checkbox"/>	 AmazonS3FullAccess	AWS managed	7
<input type="checkbox"/>	 AWSLambdaBasicExecutionRole-897acf6...	Customer managed	1

Amazon S3

Almacenamiento del Token de Google Drive

Se utiliza S3 para almacenar el token.pickle que permite acceder a la API de Google Drive y las imágenes de maquinaria en obras de construcción sobre las cuales se realizarán las inferencias. Para ello, se crea un bucket de S3 y define que sea público, tal y como se ve en las imágenes:



Info

Objects

Properties

Permissions

Metrics

Managem

Objects (1) Info



Copy S3 URI

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inve](#)

Find objects by prefix

<input type="checkbox"/>	Name	▲	Type
<input type="checkbox"/>	token.pickle		pickle

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, and IAM permissions. These settings apply only to this bucket and its access points. AWS recommends that you turn off public access. If you require some level of public access to your buckets or objects within, you can customize the settings.

Block *all* public access

 Off

► Individual Block Public Access settings for this bucket

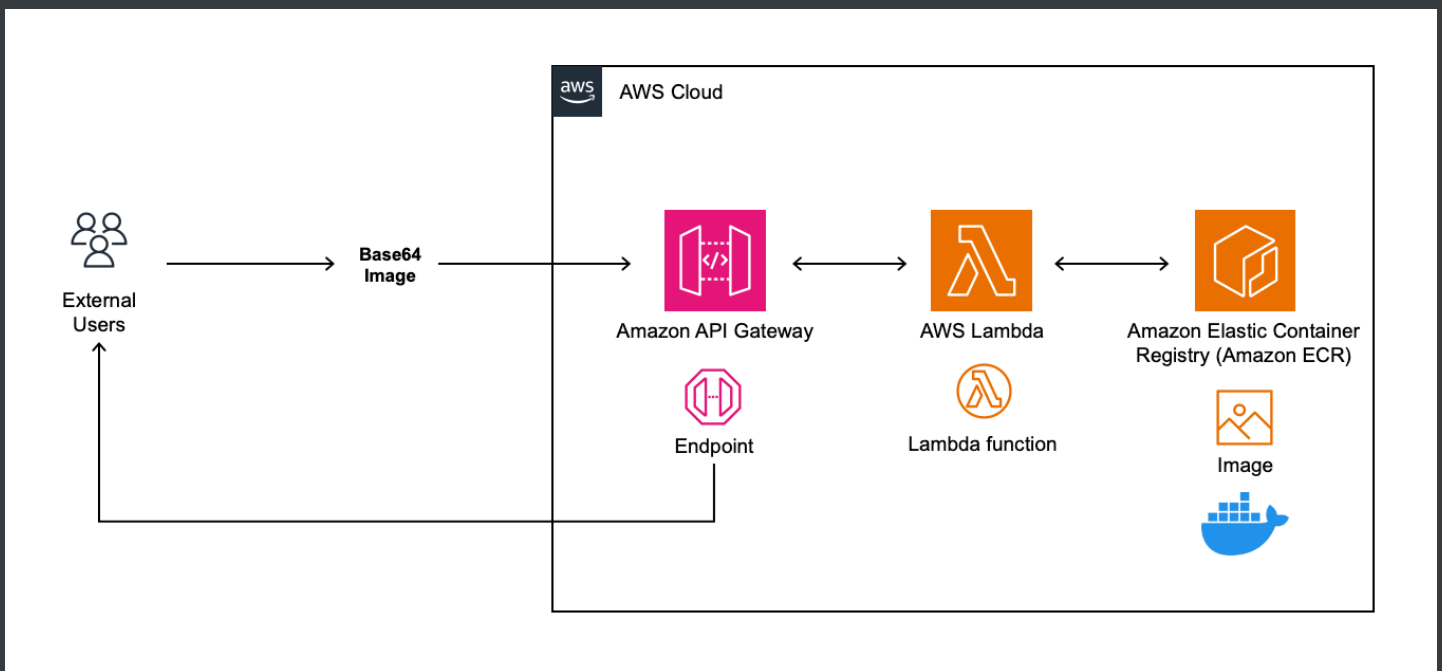
Configuración del API

La estructura del API se constituye para seguir el siguiente flujo de procesos:

1. **Envío de Imágenes por Usuarios Externos:** Los usuarios externos envían imágenes de obras de construcción a un endpoint definido dentro del API Gateway, en un formato de Base64.
2. **Amazon API Gateway:** El API Gateway recibe las imágenes de obras de construcción, las enruta y envía a la función de Lambda, para ser procesadas.
3. **Amazon Lambda:** La función de Lambda recibe las imágenes de obras de construcción. Posteriormente, utiliza el contenedor de Docker almacenado en ECR para realizar las inferencias sobre las imágenes y detectar la maquinaria en obras de construcción.
4. **Amazon Elastic Container Registry (ECR):** ECR almacena la imagen de Docker que contiene el modelo de detección de objetos y los archivos de configuración necesarios para realizar las inferencias. La imagen de Docker es utilizada por la función de Lambda para realizar las inferencias sobre las imágenes de maquinaria en obras de construcción.

5. **Procesamiento y Respuesta:** La función de Lambda procesa las imágenes de obras de construcción y detecta la maquinaria en obras de construcción. Posteriormente, envía la respuesta con las inferencias obtenidas mediante el modelo almacenado dentro del contenedor al API Gateway.
6. **Respuesta a Usuarios Externos:** El API Gateway recibe la respuesta con las inferencias obtenidas y la envía a los usuarios externos que enviaron las imágenes de obras de construcción.

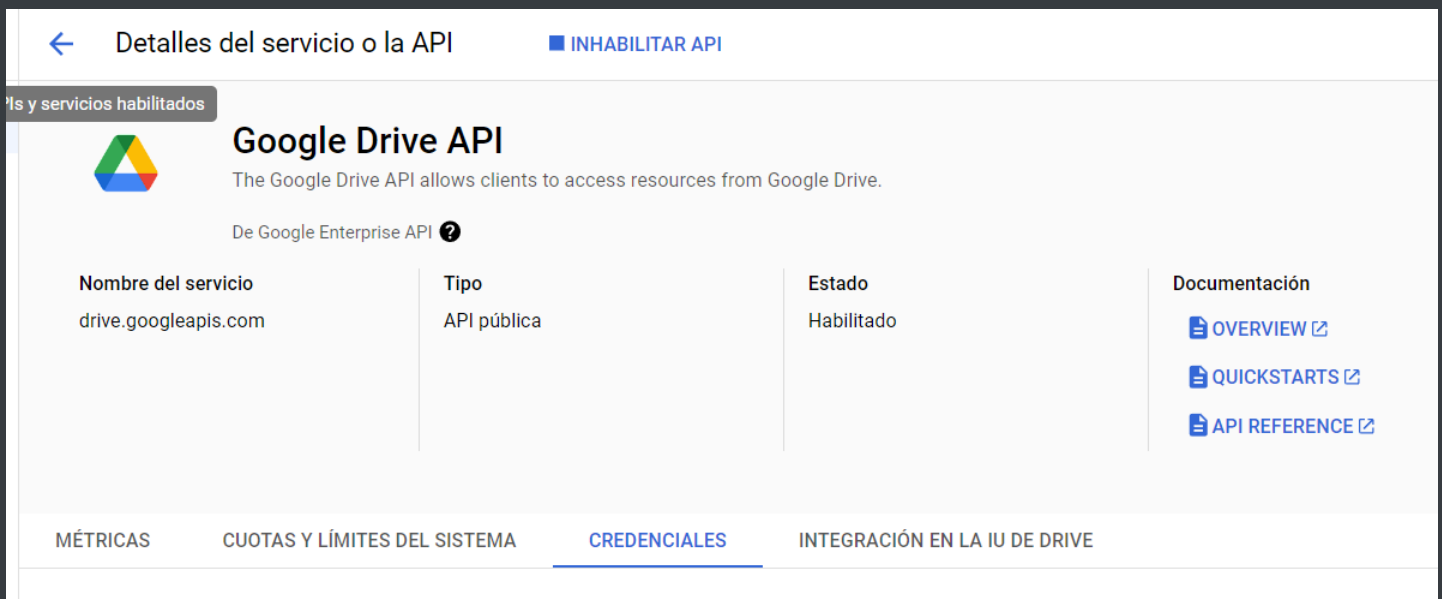
La estructura del API se muestra representada visualmente en la siguiente figura:



[Volver al inicio](#)

Configuración de Google Drive API

Para la configuración de Google Drive API, se debe ir a la consola de Google Cloud, crear un proyecto y seleccionarlo. Luego, ir a la sección de API y Servicios y habilitar la API de Google Drive para el proyecto. Posteriormente, ir a la opción de Administrar para crear credenciales para usar la API.



The screenshot shows the Google Cloud Console interface for the Google Drive API. At the top, there's a navigation bar with a back arrow, the title "Detalles del servicio o la API", and a button "INHABILITAR API". Below this, a tab "APIs y servicios habilitados" is active. The main content area features the Google Drive logo, the title "Google Drive API", and a description: "The Google Drive API allows clients to access resources from Google Drive." Below the description, it says "De Google Enterprise API" with a help icon. A table displays the service details:

Nombre del servicio	Tipo	Estado	Documentación
drive.googleapis.com	API pública	Habilitado	OVERVIEW QUICKSTARTS API REFERENCE





At the bottom, there's a navigation bar with four tabs: "MÉTRICAS", "CUOTAS Y LÍMITES DEL SISTEMA", "CREDENCIALES" (which is selected and underlined), and "INTEGRACIÓN EN LA IU DE DRIVE".

A la hora de crear las credenciales, se debe seleccionar la opción de Datos de Usuario y dar un nombre a la aplicación. Se debe ingresar el correo electrónico donde se requiera. Se debe establecer el tipo de aplicación como Aplicación de Escritorio y descargar el archivo de secretos del cliente que se proporciona en el último paso.

Credenciales compatibles con esta API



Para ver todas las credenciales, visita [Credenciales en la sección API y servicios](#)

ID de clientes OAuth 2.0

<input type="checkbox"/>	Nombre	Fecha de creación ↓	Tipo	ID de cliente	Acciones
<input type="checkbox"/>		29 abr 2024	Escritorio		   

Cuentas de servicio

[Administrar cuentas de servicio](#)

<input type="checkbox"/>	Correo electrónico	Nombre ↑	Acciones
<input type="checkbox"/>			 

Ahora que el ID de cliente de OAuth está creado, se debe crear una cuenta de servicio y conceder acceso de owner a la Service Account. Finalmente, se genera una API key y se establecen las restricciones de la API de Google Drive.

Con esto, la configuración está completa. Se selecciona una carpeta de Drive y se guarda su ID para más adelante. Se puede encontrar navegando a la carpeta y mirando la URL.

Una vez que se tiene el JSON de credenciales, se debe ejecutar el archivo `start_connection.py`, el cual se encuentra documentado a continuación.

Documentación de Archivo `Start_Connection.py`

Este código en Python implementa la autenticación y acceso a Google Drive utilizando la API de Google Drive.

```
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from google.auth.transport.requests import Request
import pickle
import os
import io

# Scope son los permisos de acceso que se dan para leer o escribir
archivos en el google drive.
SCOPES = ['https://www.googleapis.com/auth/drive']
```

```

def main():
    creds = None
    # El archivo de token.pickle es el que almacena la información del
    # usuario, este es creado automaticamente cuando
    # el proceso de autenticación sucede por primera vez.
    if os.path.exists('token.pickle'):
        pickle_data = b"token.pickle"
        byte_stream = io.BytesIO(pickle_data)
        creds = pickle.load(pickle_data)
        print("Pickle read")

    # Si el archivo token.pickle no es encontrado, este direcciona a
    # realizarse la autenticación desde la web.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow =
InstalledAppFlow.from_client_secrets_file('client_secret.json', SCOPES)
#Archivo json creado desde GCP
            creds = flow.run_local_server(port=0)
            # Save the credentials for the next run
            with open('token.pickle', 'wb') as token:
                pickle.dump(creds, token)

    service = build('drive', 'v3', credentials=creds)

    # Ejemplo para probar el acceso
    results = service.files().list(
        pageSize=10, fields="nextPageToken, files(id, name)").execute()
    items = results.get('files', [])

    if not items:

```



```
data = load_pickle_from_s3(bucket, key)
if data:
    print("Se carga el pickle")
else:
    print("Falla en carga de pickle")

'''
```

1. Importaciones:

- **google_auth_oauthlib.flow** : Importa `InstalledAppFlow` , utilizado para manejar el flujo de autenticación con OAuth 2.0.
- **googleapiclient.discovery** : Importa `build` para construir un objeto de servicio que interactúa con la API de Google.
- **google.auth.transport.requests** : Importa `Request` para manejar solicitudes HTTP durante el proceso de autenticación.
- **pickle** : Se utiliza para serializar y deserializar objetos Python, permitiendo guardar y cargar credenciales.
- **os y io** : Utilizados para manejo de archivos y flujos de bytes en memoria, respectivamente.

2. Constantes:

- **SCOPES** : Define los permisos necesarios para acceder a los archivos de Google Drive. En este caso, el permiso es para leer y escribir archivos.

3. Función `main` :

- **Manejo de credenciales:**
 - Verifica si existe un archivo `token.pickle` que contiene credenciales previamente guardadas. Si existe, las carga para evitar la necesidad de autenticarse nuevamente.
 - Si las credenciales no son válidas o han expirado, y existe un `refresh_token` , intenta actualizar las credenciales. De lo contrario, inicia el flujo de autenticación para obtener nuevas credenciales.

- **Flujo de Autenticación:**

- Utiliza `InstalledAppFlow` para iniciar un proceso de autenticación en caso de no encontrar credenciales válidas. Este proceso abrirá una ventana del navegador para que el usuario inicie sesión y autorice la aplicación.
- Guarda las nuevas credenciales en `token.pickle` para futuras ejecuciones.

- **Construcción del Servicio:**

- Crea un objeto de servicio para interactuar con la API de Google Drive, utilizando las credenciales obtenidas.

- **Listado de Archivos:**

- Utiliza el servicio para listar los primeros 10 archivos en Google Drive, mostrando su nombre y ID.
- Imprime los archivos encontrados, o un mensaje indicando que no se encontraron archivos si la lista está vacía.

4. Ejecución:

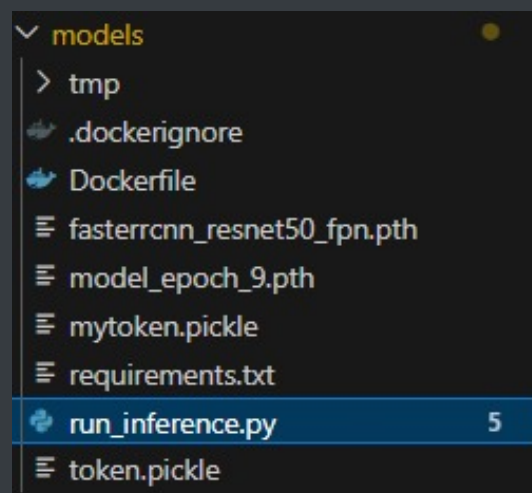
- Si este script es el archivo principal que se está ejecutando (`__name__ == '__main__'`), entonces llama a la función `main`.

[Volver al inicio](#)

Estructura del Docker

El sistema de detección de maquinaria en obras de construcción se encuentra encapsulado en una imagen de Docker. La imagen de Docker contiene el modelo de detección de objetos, los archivos de configuración necesarios para realizar las inferencias y las dependencias del sistema. La imagen de Docker se utiliza para desplegar el sistema en cualquier entorno de desarrollo o producción de forma rápida y sencilla.

La imagen de Docker se compone de los siguientes elementos:



Documentación del Código de Inferencia `run_inference.py`

Este código en Python integra varias tecnologías incluyendo PyTorch, AWS S3, Google Drive API y Pandas para realizar tareas de inferencia con un modelo de detección de objetos y gestionar la salida en la nube.

```
import os
# TORCH_HOME environment en '/tmp' es necesario para la escritura en
Lambda
os.environ['TORCH_HOME'] = '/tmp'
```

```

from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
import pickle
import os
import torch
from torchvision.models.detection import fasterrcnn_resnet50_fpn,
FasterRCNN_ResNet50_FPN_Weights
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.transforms import functional as F
from PIL import Image
from googleapiclient.discovery import build
from googleapiclient.http import MediaFileUpload
from datetime import datetime
import io
import pandas as pd
import boto3
import csv
from io import StringIO
from googleapiclient.http import MediaIoBaseUpload

                #["dump_truck", "person", "excavator", "loader",
                "mixer_truck",'steamroller']
class_mapping = ["Vagoneta", "Persona", "Tractor", "Cargador",
                "Mezcladora",'Aplanadora']
print(class_mapping)

#Definir el umbral de aceptación.
threshold = 0.7
num_classes = 7

FOLDER_ID = '1' #folderID de dIAra Account

#Función para importar el token.pickle desde S3
def load_pickle_from_s3(bucket_name, object_key):

```

```

s3_client = boto3.client('s3')
try:
    response = s3_client.get_object(Bucket=bucket_name,
Key=object_key)
    file_content = response['Body'].read()
    data_stream = io.BytesIO(file_content)
    data = pickle.load(data_stream)
    print("Pickle Cargado")
    return data
except Exception as e:
    print(f"Falla en cargar pickle desde S3: {e}")
    return None

```

#Función para guardar archivo en S3 y despues transferir a Google Drive

```

def save_csv_to_s3_movetogdrive(data):
    s3_client = boto3.client('s3')
    csv_buffer = StringIO()
    data.to_csv(csv_buffer, index=False)

    print(csv_buffer.getvalue(), "primer")
    csv_buffer.seek(0)

    today = datetime.now()
    formatted_date = today.strftime('%Y%m%d')
    formatted_date
    print(formatted_date)

    bucket_name = 'c'
    object_key = f'{formatted_date}.csv'
    s3_client.put_object(Bucket=bucket_name, Key=object_key,
Body=csv_buffer.getvalue(), ContentType='text/csv')
    print( f' Archivo Cargado satisfactoriamente {object_key} a
{bucket_name}')

```

```

#Enviar archivo de S3 a Drive
s3_client = boto3.client('s3')
s3_response = s3_client.get_object(Bucket = bucket_name, Key=
object_key)
file_data = s3_response['Body'].read()
file_stream = io.BytesIO(file_data)
key = 'token.pickle'
creds = load_pickle_from_s3(bucket_name, key)
service = build('drive', 'v3', credentials=creds)
archivos_folderID = '1' #Folder ID de Archivos dIAra

file_metadata = { 'name' : object_key, 'parents': [archivos_folderID]
}

media = MediaIoBaseUpload(file_stream,mimetype='application/octet-
stream', resumable=True)

try:
    file = service.files().create(body=file_metadata,
                                media_body=media,
                                fields='id').execute()

    if 'id' in file:
        print(f"Archivo cargado con el ID: {file['id']}")
except Exception as e:
    print(f"Error: {e}")

#Función para enlistar a todos los archivos que estan en la carpeta del
día y trae las credenciales
def list_files(folder_id):

    today = datetime.now()
    formatted_date = today.strftime('%Y%m%d')
    formatted_date
    print(formatted_date)

```

```

SCOPES = ['https://www.googleapis.com/auth/drive']

bucket = 'c'
key = 'token.pickle'
creds = load_pickle_from_s3(bucket, key)

service = build('drive', 'v3', credentials=creds)
query = f"'{folder_id}' in parents"
result = service.files().list(q=query).execute()
files = result.get('files', [])

imgs_id = []

for file in files:
    folder = file['name']
    id_folder = file['id']
    if folder.find('.') == -1:
        if folder == formatted_date:
            query = f"'{id_folder}' in parents and mimeType !=
'application/vnd.google-apps.folder'"
            result = service.files().list(q=query).execute()
            imgs = result.get('files', [])
            for img in imgs:
                if img['name'].find('.') != -1:
                    imgs_id.append([img['name'], img['id']])
            break
return imgs_id, formatted_date, creds

```

#Función para leer la arquitectura base de FasterRCNN

```

def get_model(num_classes, weights_path='./fasterrcnn_resnet50_fpn.pth'):
    model = fasterrcnn_resnet50_fpn(pretrained=False)
    model.load_state_dict(torch.load(weights_path))
    in_features = model.roi_heads.box_predictor.cls_score.in_features

```

```
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features,
num_classes)
    return model
```

#Función principal para inferir en los archivos leídos desde google drive
def main():

```
    img_inference, formatted_date, creds = list_files(FOLDER_ID)
    print(img_inference)
    if img_inference != []:
        #Cargar el modelo deseado (tomar en cuante las transformaciones)
        model = get_model(num_classes)
        model.load_state_dict(torch.load('./model_epoch_99.pth',
map_location=torch.device('cpu'))))
        model.eval()
        device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')
        model = model.to(device)

        imgs_name = []
        img_clase = []
        img_prob = []
        img_bbox = []

        #Proceso de inferencia y crear archivo de resultados
        service = build('drive', 'v3', credentials=creds)
        for img_name, img_id in img_inference:
            file_metadata =
service.files().get_media(fileId=img_id).execute()
            file = io.BytesIO(file_metadata)
            img_pil = Image.open(file)
            img_tensor = F.to_tensor(img_pil).unsqueeze(0)
            image = img_tensor.to(device)
            with torch.no_grad():
```



```

        prediction = model(image)
        bboxes = prediction[0]['boxes']
        labels = prediction[0]['labels']
        scores = prediction[0]['scores']
        for bbox, label, score in zip(bboxes, labels, scores):
            if score > threshold:
                objeto = class_mapping[int(label)-1]
                print(f"Imagen: {img_name}, se encontro: el objeto
{objeto}, bbox {bbox}, score = {score}")
                imgs_name.append(img_name)
                img_clase.append(objeto)
                img_prob.append(score)
                img_bbox.append(bbox)

    data = pd.DataFrame()
    data["nombre_imagen"] = imgs_name
    data["clase"] = img_clase
    data["probabilidad"] = img_prob
    data["recuadro"] = img_bbox

    save_csv_to_s3_movetogdrive(data)

if __name__ == '__main__':
    main()

```

1. Importaciones y Configuraciones Iniciales:

- Importa las bibliotecas necesarias para trabajar con archivos, modelos de aprendizaje automático, manejo de imágenes, acceso a Google Drive, y almacenamiento en AWS S3.
- Establece la ruta para la caché de modelos de PyTorch usando `os.environ['TORCH_HOME']`.

2. Variables y Funciones:

- **class_mapping** : Un arreglo que mapea índices a nombres de clases para las detecciones del modelo.
- **threshold, num_classes, FOLDER_ID** : Variables para configurar el umbral de detección, el número de clases y el ID del folder en Google Drive.

3. Funciones de Manejo de Datos en S3 y Google Drive:

- **load_pickle_from_s3** : Carga un archivo pickle desde un bucket de S3, utilizado para manejar credenciales de autenticación.
- **save_csv_to_s3_movetogdrive** : Guarda un DataFrame de Pandas como un archivo CSV en S3 y luego lo transfiere a Google Drive.
- **list_files** : Lista archivos en una carpeta específica de Google Drive y carga credenciales desde S3.

4. Función para Cargar y Configurar el Modelo de Detección:

- **get_model** : Carga y configura un modelo de detección de objetos Fasterrcnn ResNet50, ajustando el predictor de clases según el número de clases proporcionado.

5. Función Principal **main** :

- Realiza una lista de imágenes a inferir desde una carpeta específica en Google Drive.
- Carga y configura el modelo para realizar inferencias sobre las imágenes listadas.
- Para cada imagen:
 - Descarga la imagen desde Google Drive.
 - Realiza la inferencia utilizando el modelo de PyTorch.
 - Guarda los resultados de la inferencia (clase, probabilidad, y bbox) en listas.
- Crea un DataFrame con los resultados y lo guarda en S3, transfiriéndolo luego a Google Drive.

6. Ejecución:

- Si el script se ejecuta como el archivo principal, se invoca la función **main** .

Documentación del Docker File

El `Dockerfile` es un archivo de configuración que contiene las instrucciones necesarias para construir la imagen de Docker. Se especifica la imagen base, las dependencias del sistema, y los archivos de configuración necesarios para realizar las inferencias usando PyTorch con soporte para GPU.

```
FROM pytorch/pytorch:2.2.1-cuda12.1-cudnn8-runtime
```

```
WORKDIR /app
```

```
COPY . /app
```

```
RUN pip install -r requirements.txt
```

```
CMD python3 run_inference.py
```

1. `FROM pytorch/pytorch:2.2.1-cuda12.1-cudnn8-runtime` :

Esta línea especifica la imagen base que se usará para construir la nueva imagen Docker. La imagen base es una versión oficial de PyTorch preconfigurada con CUDA 12.1 y cuDNN 8 para soporte de GPU. Esto facilita el desarrollo y despliegue de aplicaciones de PyTorch que requieren aceleración por GPU.

2. `WORKDIR /app` :

Establece el directorio de trabajo dentro del contenedor Docker. Todos los comandos que se ejecuten posteriormente en el `Dockerfile` se realizarán en este directorio `/app`.

3. `COPY . /app` :

Copia todos los archivos del directorio actual en la máquina host (donde se está construyendo la imagen Docker) al directorio `/app` dentro del contenedor. Esto incluye todos los scripts, archivos de código, y otros recursos necesarios para ejecutar la aplicación.

4. `RUN pip install -r requirements.txt` :

Ejecuta el comando `pip install` para instalar las dependencias de Python listadas en el archivo `requirements.txt` . Esto asegura que todas las librerías necesarias estén disponibles en el contenedor, permitiendo que el script de Python se ejecute correctamente.

5. CMD `python3 run_inference.py` :

Define el comando por defecto que se ejecutará cuando el contenedor Docker sea iniciado. En este caso, ejecutará el script `run_inference.py` utilizando Python 3. Este script probablemente contiene el código para realizar la inferencia utilizando el modelo de PyTorch configurado.

`fasterrcnn_resnet50_fpn.pth`

Contiene el modelo de detección de objetos pre-entrenado en un conjunto de datos de imágenes base. El modelo de detección de objetos se basa en FasterRCNN con ResNet50 como red base.

`model_epoch_9.pth`

Contiene el modelo de detección de objetos adaptado y afinado con un conjunto de datos de imágenes de maquinaria en obras de construcción, para cumplir los requisitos específicos del proyecto.

`requirements.txt`

Contiene las dependencias del sistema necesarias para ejecutar el modelo de detección de objetos. Se instalan automáticamente al construir la imagen de Docker.

`run_inference.py`

Es el script principal que se encarga de realizar las inferencias sobre las imágenes de maquinaria en obras de construcción. El script carga el modelo de detección de objetos, procesa las imágenes de entrada y genera las predicciones de los objetos detectados.

`token.pickle`

Contiene las credenciales de acceso a la API de Google Drive. El token.pickle se utiliza para autenticar el sistema y acceder a las imágenes de maquinaria en obras de construcción almacenadas en Google Drive.

[Volver al inicio](#)