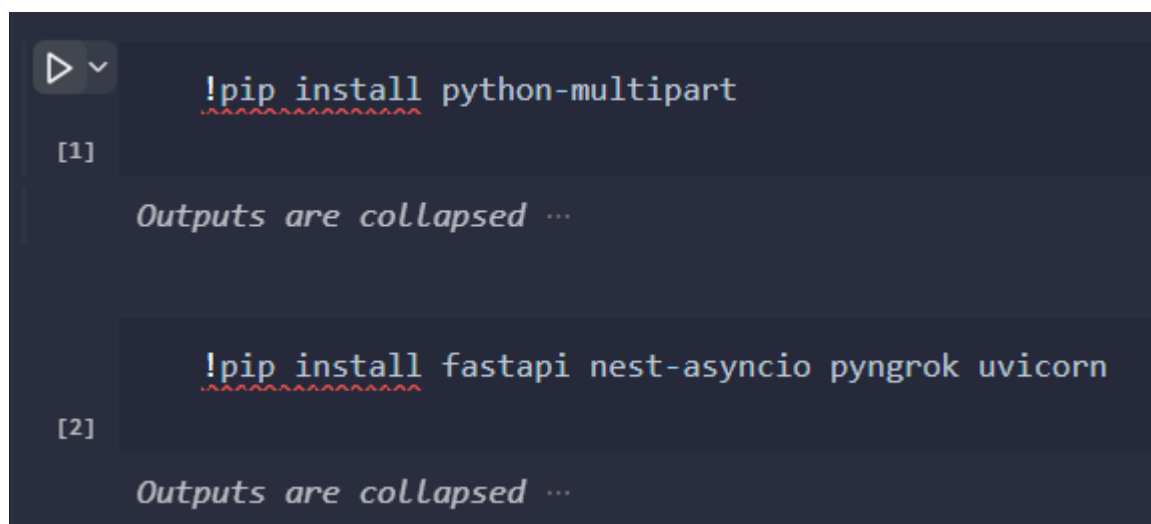


Instructivo para el despliegue y uso de la aplicación

Junto con este documento se encuentra una carpeta llamada “AgroDiagnosticolA”, dentro de esta carpeta se encuentra un archivo .jpynb llamado “Despliegue”, abrir este archivo con visual studio, o en su defecto usar google collab (para este caso habrá que cargar todos los archivos y carpetas a colab además de modificar ciertas rutas por lo que no se recomienda). Una vez abierto el archivo se prosigue con los siguientes pasos:

Paso 1:

Realizamos las instalaciones necesarias ejecutando las siguientes celdas.



```
[1] !pip install python-multipart
```

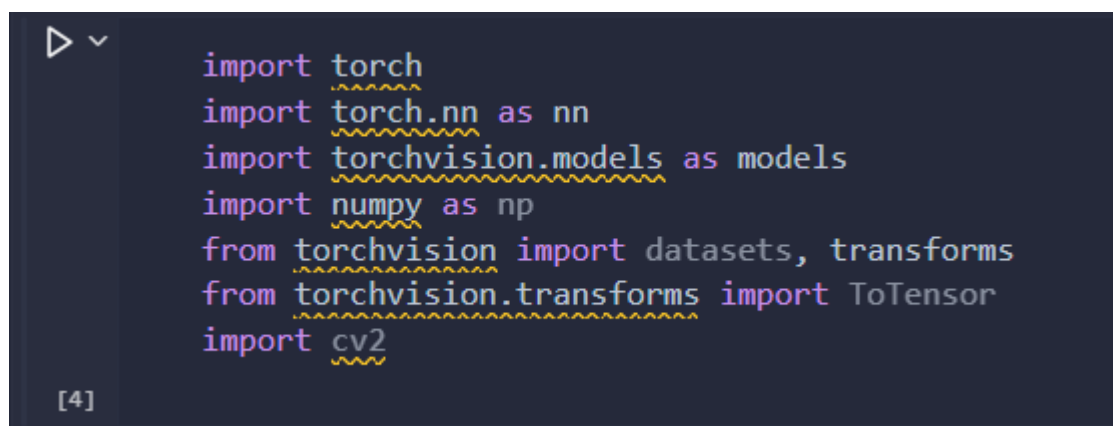
Outputs are collapsed ...

```
[2] !pip install fastapi nest-asyncio pyngrok uvicorn
```

Outputs are collapsed ...

Paso 2:

Se ejecutan las siguientes celdas para cargar el modelo ya entrenado.



```
[4] import torch
import torch.nn as nn
import torchvision.models as models
import numpy as np
from torchvision import datasets, transforms
from torchvision.transforms import ToTensor
import cv2
```

```

# Función para entrenar una red neuronal convolucional basada en el modelo ResNet

class ResNetModel(nn.Module):
    def __init__(self):
        super(ResNetModel, self).__init__()
        # Cargar el modelo ResNet preentrenado
        self.resnet = models.resnet50(pretrained=True)

        # Reemplazar la capa final de clasificación de ResNet para adaptarse al número de clases deseado
        in_features = self.resnet.fc.in_features
        self.resnet.fc = nn.Identity() # Remover la capa final existente

        # Definir las nuevas capas de clasificación
        self.fc1 = nn.Linear(in_features, 32)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(32, 4)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.resnet(x) # Pasar por el ResNet sin la capa final
        x = self.fc1(x)    # Pasar por la capa densa 1
        x = self.relu(x)   # Aplicar ReLU
        x = self.fc2(x)    # Pasar por la capa densa 2
        x = self.softmax(x) # Aplicar Softmax
        return x

```

```

device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)

model = ResNetModel().to(device)

# Ruta al archivo checkpoint
ruta_checkpoint = 'Modelo.pt'

# Cargar el estado del modelo desde el checkpoint
model.load_state_dict(torch.load(ruta_checkpoint, map_location=device))

```

Paso 3:

Se ejecutan las celdas con los importes y definiciones necesarias para usar fastAPI.

```
from fastapi import FastAPI, Request, Form, File, UploadFile
from fastapi.templating import Jinja2Templates
from fastapi.staticfiles import StaticFiles
from fastapi.responses import RedirectResponse
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import HTMLResponse
import cv2
import numpy as np
import io
from fastapi.responses import StreamingResponse
import base64
```

```
# Definir las transformaciones de preprocesamiento
def preprocess_image(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Cambia de BGR a RGB
    image = image.astype(np.float32) / 255.0 # Normaliza a [0, 1]
    tensor_image = torch.tensor(image).permute(2, 0, 1) # Cambia la forma a CxHxW
    tensor_image = tensor_image.unsqueeze(0) # Añade la dimensión del batch
    return tensor_image

def resize_image(img):
    # Convierte a un array NumPy desde el contenido de la imagen
    nparr = np.frombuffer(img, np.uint8)
    image = cv2.imdecode(nparr, cv2.IMREAD_COLOR) # Decodifica la imagen

    if image is None:
        raise ValueError("Error al decodificar la imagen.")

    # Verifica la forma de la imagen
    shape = image.shape

    # Redimensionar si no tiene el tamaño esperado
    if shape[0] == 224 and shape[1] == 224:
        return image
    else:
        dim = (224, 224)
        resized = cv2.resize(image, dim)
        return resized

def predict(image_tensor):
    with torch.no_grad(): #desactivamos el calculo de gradientes
        output = model(image_tensor) #hacer la inferencia
        _, predicted = torch.max(output.data, 1) #obtener la clase predicha
    return predicted.item()
```

```

app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=['*'],
    allow_credentials=True,
    allow_methods=['*'],
    allow_headers=['*'],
)

# Montar archivos estáticos (CSS, imágenes, etc.)
app.mount("/static/css", StaticFiles(directory = "Templates/StylesCSS"), name="css")
app.mount("/static/images", StaticFiles(directory = "Templates/images"), name="images")
app.mount("/static/js", StaticFiles(directory = "Templates"), name="js")

# Configurar las plantillas
templates = Jinja2Templates(directory="Templates/")

@app.get("/")
async def landing_page(request: Request):
    return templates.TemplateResponse("index.html", {"request": request, "title": "Agro Diagnóstico IA"})

@app.get("/upload/")
async def upload_page(request: Request):
    return templates.TemplateResponse("plataforma.html", {"request": request})

@app.post("/upload/")
async def handle_upload(request: Request, file: UploadFile = File(...)):
    try:
        content = await file.read() # Lee el contenido del archivo
        nparr = np.frombuffer(content, np.uint8)
        image = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

```

Paso 4:

Ejecutar la celda de despliegue de la aplicación

```

import nest_asyncio
from pyngrok import ngrok
import uvicorn

# Get your authtoken from https://dashboard.ngrok.com/get-started/your-authtoken
auth_token = "2mlWw19c2JDSbJp2DF10H1zqYPy_4QUs4RZMwQJoERjDM5HHp"

# Set the authtoken
ngrok.set_auth_token(auth_token)

# Connect to ngrok
ngrok_tunnel = ngrok.connect(8000)

# Print the public URL
print('Public URL:', ngrok_tunnel.public_url)

# Apply nest_asyncio
nest_asyncio.apply()

# Run the uvicorn server
uvicorn.run(app, port=8000)

```

Paso 5:

Una vez ejecutada, esta celda permanecerá activa mientras no se detenga manualmente. Para acceder al servicio dar click en el enlace llamado “Public URL”

```
... INFO: Started server process [2540]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
Public URL: https://f674-2800-e2-1a00-8dad-655f-5f7d-224c-6b4e.ngrok-free.app
```

Paso 6:

Al ingresar en el enlace, se abrirá el navegador mostrando el siguiente recuadro:

You are about to visit:

f674-2800-e2-1a00-8dad-655f-5f7d-224c-6b4e.ngrok-free.app

Website IP: 2800:e2:1a00:8dad:655f:5f7d:224c:6b4e

- This website is served for free through ngrok.com.
- You should only visit this website if you trust whoever sent the link to you.
- Be careful about disclosing personal or financial information like passwords, phone numbers, or credit cards.

[Visit Site](#)

Dar click al botón Visit Site para continuar.

Paso 7:

Una vez dentro dar click donde dice plataforma para acceder al lugar donde se suben las imágenes para recibir un diagnóstico.



[Plataforma](#) [Acerca de](#) [Blog](#) [Contacto](#) [REGISTRATE](#) [INICIAR SESIÓN](#)

Protege tus cultivos y asegura tu producción con ayuda de Inteligencia Artificial

Usa la **Inteligencia Artificial** para saber lo que necesitan tus cultivos. Automatiza con nosotros el proceso de detección y diagnóstico de plagas y enfermedades.



¿Cómo puedes utilizar nuestro servicio?

Nota: debido a que la página no está hecha para ser web responsive puede que al ejecutarse algunos elementos se desordenen.

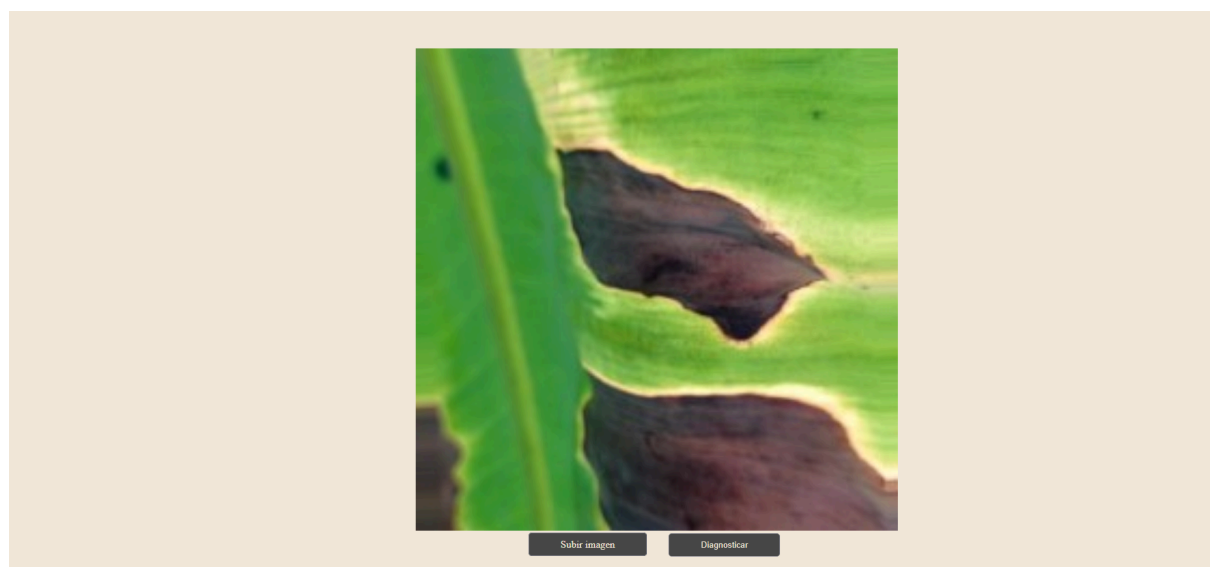
Paso 8:

Una vez dentro de la plataforma, desplazarse hasta el final en donde se encontrarán los botones para subir tu imagen.



Paso 9:

Dar click al botón "Subir imagen" y selecciona una imagen (.jpg) de tu equipo (se adjuntan algunas imágenes de prueba). Se mostrará una previsualización en pantalla.



Paso 10:

Da click en el botón "Diagnosticar" y recibe el diagnóstico. Para realizar otro diagnóstico repite el paso 9.

Nota: En caso de que el despliegue no funcione contactar al correo de alguien del equipo.