```python
import gradio as gr
import torch
import numpy as np
import shutil
import os
import torch
import TwinLite as net
from PIL import Image
from dotenv import load_dotenv
load_dotenv()

share = os.getenv("SHARE", False)


model = net.TwinLiteNet()
import cv2

def Run(model,img):
    img = cv2.resize(img, (640, 360))
    img_rs=img.copy()

    img = img[:, :, ::-1].transpose(2, 0, 1)
    img = np.ascontiguousarray(img)
    img=torch.from_numpy(img)
    img = torch.unsqueeze(img, 0)  # add a batch dimension
    img=img.float() / 255.0
    img = img
    with torch.no_grad():
        img_out = model(img)
    x0=img_out[0]
    x1=img_out[1]

    _,da_predict=torch.max(x0, 1)
    _,ll_predict=torch.max(x1, 1)

    DA = da_predict.byte().cpu().data.numpy()[0]*255
    LL = ll_predict.byte().cpu().data.numpy()[0]*255
    img_rs[DA>100]=[255,0,0]
    img_rs[LL>100]=[0,255,0]

    return img_rs


model = net.TwinLiteNet()
model = torch.nn.DataParallel(model)
model.load_state_dict(torch.load('fine-tuned-model.pth', map_location=torch.device('cpu')))
model.eval()


def predict(image):
    image = Image.fromarray(image.astype('uint8'), 'RGB')
    image.save("input.png")
    img = cv2.imread("input.png")
    img = Run(model, img)
    cv2.imwrite("sample.png", img)
    prediction = Image.open("sample.png")
    return prediction


iface = gr.Interface(fn=predict, inputs="image", outputs="image", title="Image Segmentation")

if __name__ == "__main__":
    if share:
        server = "0.0.0.0"
    else:
        server = "127.0.0.1"
    iface.launch(server_name = server)
```