import torch

import torch.nn as nn

from torchvision.models import efficientnet\_b0, EfficientNet\_B0\_Weights

from pathlib import Path

from PIL import Image

from torchvision import transforms as T

# Mapeo de clases

KEEP\_CLASSES = [0, 1, 2, 5, 6]

OLD2NEW = {old: new for new, old in enumerate(KEEP\_CLASSES)}

NEW2OLD = {new: old for old, new in OLD2NEW.items()}

# Definición del modelo

class ImageClassifier(nn.Module):

def \_\_init\_\_(self, meta\_dim: int, num\_classes: int, pretrained: bool = False, dropout: float = 0.5):

super().\_\_init\_\_()

weights = EfficientNet\_B0\_Weights.IMAGENET1K\_V1 if pretrained else None

self.backbone = efficientnet\_b0(weights=weights)

in\_feats = self.backbone.classifier[1].in\_features

self.backbone.classifier = nn.Identity()

self.meta = nn.Sequential(

nn.Linear(meta\_dim, 32),

nn.ReLU(inplace=True),

nn.BatchNorm1d(32),

)

self.head = nn.Sequential(

nn.Linear(in\_feats + 32, 256),

nn.ReLU(inplace=True),

nn.Dropout(dropout),

nn.Linear(256, num\_classes)

)

def forward(self, x\_img, x\_meta):

f\_img = self.backbone(x\_img)

f\_meta = self.meta(x\_meta)

f = torch.cat([f\_img, f\_meta], dim=1)

return self.head(f)

class OcularPredictor:

def \_\_init\_\_(self, model\_path: Path):

self.device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

self.model = self.\_load\_model(model\_path)

self.transform = self.\_get\_transforms()

self.gender\_map = {"male": 1, "female": 0}

def \_load\_model(self, model\_path: Path):

"""Carga el modelo y sus pesos."""

model = ImageClassifier(

meta\_dim=2,

num\_classes=len(KEEP\_CLASSES),

pretrained=False

).to(self.device)

model.load\_state\_dict(torch.load(str(model\_path), map\_location=self.device))

model.eval()

return model

def \_get\_transforms(self):

"""Devuelve las transformaciones para la imagen."""

return T.Compose([

T.ToTensor(),

T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

def preprocess(self, image\_path: str, meta\_data: dict):

"""

Preprocesa la imagen y los metadatos para la predicción.

"""

# 1. Preprocesar la imagen

image\_path = Path(image\_path)

if not image\_path.is\_file():

raise FileNotFoundError(f"El archivo de imagen no existe: {image\_path}")

image = Image.open(image\_path).convert('RGB')

image\_tensor = self.transform(image).unsqueeze(0).to(self.device)

# 2. Preprocesar los metadatos

meta\_tensor = torch.tensor(

[meta\_data["age"], self.gender\_map[meta\_data["gender"]]],

dtype=torch.float32

).unsqueeze(0).to(self.device)

return image\_tensor, meta\_tensor

def predict(self, image\_path: str, meta\_data: dict):

"""

Realiza una predicción sobre una imagen y metadatos.

Args:

image\_path: Ruta a la imagen.

meta\_data: Diccionario con los metadatos, ej: {"age": 45, "gender": "male"}

Returns:

Tupla con la clase original predicha y las probabilidades.

"""

image\_tensor, meta\_tensor = self.preprocess(image\_path, meta\_data)

# 3. Realizar la predicción

with torch.no\_grad():

logits = self.model(image\_tensor, meta\_tensor)

probabilities = torch.nn.functional.softmax(logits, dim=1)

# Obtener el índice de la clase con la probabilidad más alta

predicted\_class\_index = torch.argmax(probabilities, dim=1).item()

original\_class\_index = NEW2OLD[predicted\_class\_index]

return original\_class\_index, probabilities[0].cpu().numpy().tolist()

# Crear una única instancia del predictor

from pathlib import Path

# Obtiene la ruta del directorio actual donde se está ejecutando el script

BASE\_DIR = Path(\_\_file\_\_).resolve().parent

# Define la ruta al modelo de forma relativa

MODEL\_PATH = BASE\_DIR / "bestmodel" / "best\_model.pt"

# Crea la instancia del predictor usando la ruta correcta

predictor = OcularPredictor(model\_path=MODEL\_PATH)