import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import random

import torch

import torch.nn as nn

import torch.nn.functional as F

from torchvision import transforms

from torch.utils.data.sampler import SubsetRandomSampler

from torch.utils.data import DataLoader

import os

from tensorflow.keras.applications import Xception

from xception import Xception

from EmbryoDataset import EmbryoDataset

df\_train = pd.read\_csv('train.csv')

df\_train.head()

# define Seaborn color palette to use

palette\_color = sns.color\_palette('pastel')

label\_counts = df\_train['Class'].value\_counts()

# plotting data on chart

plt.figure(figsize=(5, 5))

plt.pie(label\_counts, labels=label\_counts.index, colors=[palette\_color[0], palette\_color[3]], explode=[0, 0.1], autopct='%1.1f%%', textprops={'fontsize': 14, 'fontweight': 'bold'})

plt.title('Percentage of Each Label', fontsize=20, fontweight='bold')

plt.axis('equal')

plt.show()

class\_weight = 1 - (label\_counts / df\_train.shape[0])

class\_weight = list(class\_weight)

df\_test = pd.read\_csv('test.csv')

df\_test.head()

# Filter images based on class

good\_images = df\_train[df\_train['Class'] == 1]['Image'].tolist()

not\_good\_images = df\_train[df\_train['Class'] == 0]['Image'].tolist()

# Randomly select one image from each class

random\_good\_image = random.choice(good\_images)

random\_not\_good\_image = random.choice(not\_good\_images)

random\_good\_image = './train/' + random\_good\_image

random\_not\_good\_image = './train/' + random\_not\_good\_image

# Plotting randomly selected images

fig, axes = plt.subplots(1, 2)

# Plot random good image

img = plt.imread(random\_good\_image)

axes[0].imshow(img)

axes[0].set\_title('Good Embryo')

axes[0].set\_xticks([])  # Remove xticks

axes[0].set\_yticks([])  # Remove yticks

# Plot random not good image

img = plt.imread(random\_not\_good\_image)

axes[1].imshow(img)

axes[1].set\_title('Not Good Embryo')

axes[1].set\_xticks([])  # Remove xticks

axes[1].set\_yticks([])  # Remove yticks

plt.tight\_layout()

plt.show()

plt.savefig('embryo\_images.png')

train\_data\_path = 'train'

test\_data\_path = 'test'

# create the train\_image\_label\_mapping

train\_image\_label\_mapping = {}

for filename in os.listdir(train\_data\_path):

    train\_image\_label\_mapping[train\_data\_path + '/' + filename] = df\_train[df\_train['Image'] == filename]['Class'].values[0]

# create the test\_image\_label\_mapping

test\_image\_label\_mapping = {}

for filename in os.listdir(test\_data\_path):

    test\_image\_label\_mapping[test\_data\_path + '/' + filename] = df\_test[df\_test['Image'] == filename]['ID'].values[0]

# defining the tranformations for the input images

transform = transforms.Compose([

    transforms.Resize((224, 224)),

    transforms.ToTensor(),

    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),

])

# set up the dataset

train\_data = EmbryoDataset(train\_image\_label\_mapping, transform=transform)

test\_data = EmbryoDataset(test\_image\_label\_mapping, train=False, transform=transform)

# Spliting train data to train and validation parts

valid\_size = 0.2

dataset\_size = len(train\_data)

indices = list(range(dataset\_size))

split = int(np.floor(valid\_size \* dataset\_size))

train\_indices, val\_indices = indices[split:], indices[:split]

train\_sampler = SubsetRandomSampler(train\_indices)

valid\_sampler = SubsetRandomSampler(val\_indices)

# Generating dataloaders

train\_loader = DataLoader(train\_data, batch\_size=4, num\_workers=2, sampler=train\_sampler, pin\_memory=True)

valid\_loader = DataLoader(train\_data, batch\_size=4, num\_workers=2, sampler=valid\_sampler, pin\_memory=True)

# test dataloader

test\_loader = DataLoader(test\_data, shuffle=False, num\_workers=2)

# Get cpu or gpu

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

# print(f'Using {device} device')

# define the model and use pretrained weights

xception = Xception()

xception.load\_state\_dict(torch.load('xception-43020ad28.pth'))

# change the number of output classes

num\_ftrs = xception.fc.in\_features

xception.fc = nn.Linear(num\_ftrs, 2)

xception = xception.to(device)

# print(xception)

criterion = nn.CrossEntropyLoss(weight=torch.Tensor(class\_weight)).cuda() if torch.cuda.is\_available() else nn.CrossEntropyLoss(weight=torch.Tensor(class\_weight))

optimizer = torch.optim.SGD(xception.parameters(), lr=1e-3, weight\_decay=1e-5)

def train\_one\_epoch(dataloader, model, criterion, optimizer):

    train\_loss = 0.0

    model.train()

    for batch, (X, y) in enumerate(dataloader):

        X, y = X.to(device), y.to(device)

        # Compute prediction error

        pred = model(X)

        loss = criterion(pred, y)

        # Backpropagation

        loss.backward()

        optimizer.step()

        optimizer.zero\_grad()

        train\_loss += loss.item()

    return train\_loss

epochs = 10

min\_valid\_loss = np.inf

for epoch in range(epochs):

    train\_loss = train\_one\_epoch(train\_loader, xception, criterion, optimizer)

    valid\_loss = 0.0

    xception.eval()

    with torch.no\_grad():

        for X, y in valid\_loader:

            X, y = X.to(device), y.to(device)

            pred = xception(X)

            loss = criterion(pred, y)

            valid\_loss += loss.item()

    print(f'Epoch {epoch+1} \t\t Training Loss: {train\_loss / len(train\_loader.sampler)} \t\t Validation Loss: {valid\_loss / len(valid\_loader.sampler)}')

    if min\_valid\_loss > valid\_loss:

        min\_valid\_loss = valid\_loss

        print('Saving...')

        state = {

                'net': xception.state\_dict(),

            }

        if not os.path.isdir('checkpoints'):

            os.mkdir('checkpoints')

        torch.save(state, 'ckpt.pth')

# define the model

best\_model = Xception(num\_classes=2)

best\_model.load\_state\_dict(torch.load('./ckpt.pth')['net'])

best\_model = best\_model.to(device)

preds = {}

best\_model.eval()

with torch.no\_grad():

    for X, id in test\_loader:

        X = X.to(device)

        pred = best\_model(X).argmax(1)

        preds[id.item()] = pred.item()

df\_preds = pd.DataFrame.from\_dict(preds, orient='index', columns=['Class']).sort\_index().reset\_index().rename(columns={'index': 'ID'})

df\_preds