ДОДАТОК А

Код програми

```
import os
import torch
from monai.transforms import (
    Compose, LoadImage, AddChannel, ScaleIntensity, ToTensor
)
from monai.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from transformers import ViTForImageClassification, ViTFeatureExtractor
from torch.optim import Adam
from torch.nn import CrossEntropyLoss
from sklearn.metrics import accuracy_score

# Підготовка до завантаження даних
class CustomDataset(Dataset):
    def __init__(self, image_files, labels, transforms):
        self.image_files = image_files
        self.labels = labels
        self.transforms = transforms

    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        image = self.transforms(self.image_files[idx])
```

```
        label = self.labels[idx]
        return image, label

# Вказання шляхів до зображень та міток
image_files = [os.path.join("path_to_tcia_data", fname) for fname in os.listdir("path_to_tcia_data")]
labels = [...]  # Відповідні мітки для кожного зображення

# Поділ на навчальні та тестові набори
train_files, test_files, train_labels, test_labels = train_test_split(image_files, labels, test_size=0.2, random_state=42)

# Визначення перетворень
transforms = Compose([LoadImage(image_only=True), AddChannel(), ScaleIntensity(), ToTensor()])

# Створення датасетів та завантажувачів
train_dataset = CustomDataset(train_files, train_labels, transforms)
test_dataset = CustomDataset(test_files, test_labels, transforms)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Завантаження екстрактора ознак та моделі
feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-base-patch16-224')
model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-224', num_labels=2)  # Кількість міток залежить від задачі

# Визначення оптимізатора та функції втрат
optimizer = Adam(model.parameters(), lr=5e-5)
criterion = CrossEntropyLoss()

# Навчання моделі
```

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

num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)


        # Перетворення вхідних даних за допомогою екстрактора ознак
        inputs = feature_extractor(images=inputs, return_tensors="pt").pixel_values.to(device)


        # Forward pass
        outputs = model(inputs).logits
        loss = criterion(outputs, labels)


        # Backward pass і оптимізація
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()


        running_loss += loss.item()


    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {running_loss / len(train_loader):.4f}')
```

```
# Оцінка моделі
model.eval()
preds, true_labels = [], []
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        inputs = feature_extractor(images=inputs, return_tensors="pt").pixel_values.to(device)
        outputs = model(inputs).logits
        _, predicted = torch.max(outputs, 1)
        preds.extend(predicted.cpu().numpy())
        true_labels.extend(labels.cpu().numpy())

accuracy = accuracy_score(true_labels, preds)
print(f'Accuracy: {accuracy:.4f}')

import os
import torch
from monai.transforms import (
    Compose, LoadImage, AddChannel, ScaleIntensity, ToTensor
)
from monai.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from transformers import DeiTForImageClassification, DeiTFeatureExtractor
from torch.optim import Adam
from torch.nn import CrossEntropyLoss
from sklearn.metrics import accuracy_score
```

```
# Підготовка до завантаження даних
class CustomDataset(Dataset):
    def __init__(self, image_files, labels, transforms):
        self.image_files = image_files
        self.labels = labels
        self.transforms = transforms

    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        image = self.transforms(self.image_files[idx])
        label = self.labels[idx]
        return image, label

# Вказання шляхів до зображень та міток
image_files = [os.path.join("path_to_tcia_data", fname) for fname in os.listdir("path_to_tcia_data")]
labels = [...]  # Відповідні мітки для кожного зображення

# Поділ на навчальні та тестові набори
train_files, test_files, train_labels, test_labels = train_test_split(image_files, labels, test_size=0.2, random_state=42)

# Визначення перетворень
transforms = Compose([LoadImage(image_only=True), AddChannel(), ScaleIntensity(), ToTensor()])

# Створення датасетів та завантажувачів
train_dataset = CustomDataset(train_files, train_labels, transforms)
```

```
test_dataset = CustomDataset(test_files, test_labels, transforms)

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)

test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Завантаження екстрактора ознак та моделі

feature_extractor = DeiTFeatureExtractor.from_pretrained('facebook/deit-base-distilled-patch16-224')

model = DeiTForImageClassification.from_pretrained('facebook/deit-base-distilled-patch16-224', num_labels=2) # Кількість міток залежить від задачі

# Визначення оптимізатора та функції втрат

optimizer = Adam(model.parameters(), lr=5e-5)

criterion = CrossEntropyLoss()

# Навчання моделі

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

model.to(device)

num_epochs = 10

for epoch in range(num_epochs):

    model.train()

    running_loss = 0.0

    for inputs, labels in train_loader:

        inputs, labels = inputs.to(device), labels.to(device)


        # Перетворення вхідних даних за допомогою екстрактора ознак

        inputs = feature_extractor(images=inputs, return_tensors="pt").pixel_values.to(device)
```

```
        # Forward pass
        outputs = model(inputs).logits
        loss = criterion(outputs, labels)

        # Backward pass і оптимізація
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {running_loss / len(train_loader):.4f}')

# Оцінка моделі
model.eval()
preds, true_labels = [], []
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        inputs = feature_extractor(images=inputs, return_tensors="pt").pixel_values.to(device)
        outputs = model(inputs).logits
        _, predicted = torch.max(outputs, 1)
        preds.extend(predicted.cpu().numpy())
        true_labels.extend(labels.cpu().numpy())

accuracy = accuracy_score(true_labels, preds)
print(f'Accuracy: {accuracy:.4f}')
```

```
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from efficientnet_pytorch import EfficientNet

# Завантаження даних
# Замість 'path_to_tcga_data.csv' використовуйте шлях до ваших геномних даних
df = pd.read_csv('path_to_tcga_data.csv')
X = df.iloc[:, :-1].values  # Геномні ознаки
y = df.iloc[:, -1].values   # Мітки

# Стандартизація даних
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Поділ на навчальну та тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Перетворення на тензори
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)
```

```
# Визначення моделі
class GenomicEfficientNet(nn.Module):
    def __init__(self):
        super(GenomicEfficientNet, self).__init__()
        self.efficientnet = EfficientNet.from_name('efficientnet-b0')
        self.fc = nn.Linear(1000, 2)  # Кількість виходів залежить від задачі

    def forward(self, x):
        x = self.efficientnet.extract_features(x)
        x = x.mean([2, 3])  # Глобальний пулінг
        x = self.fc(x)
        return x

model = GenomicEfficientNet()

# Визначення оптимізатора та функції втрат
optimizer = optim.Adam(model.parameters(), lr=1e-3)
criterion = nn.CrossEntropyLoss()

# Навчання моделі
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    optimizer.zero_grad()
```

```
    # Forward pass
    inputs = X_train.unsqueeze(1).to(device)
    labels = y_train.to(device)
    outputs = model(inputs)
    loss = criterion(outputs, labels)

    # Backward pass і оптимізація
    loss.backward()
    optimizer.step()

    running_loss += loss.item()
    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {running_loss:.4f}')

# Оцінка моделі
model.eval()
with torch.no_grad():
    inputs = X_test.unsqueeze(1).to(device)
    labels = y_test.to(device)
    outputs = model(inputs)
    _, predicted = torch.max(outputs, 1)
    accuracy = accuracy_score(labels.cpu(), predicted.cpu())
    print(f'Accuracy: {accuracy:.4f}')

import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score

from swin_transformer_pytorch import SwinTransformer

# Завантаження даних

# Замість 'path_to_tcga_data.csv' використовуйте шлях до ваших геномних даних

df = pd.read_csv('path_to_tcga_data.csv')

X = df.iloc[:, :-1].values  # Геномні ознаки

y = df.iloc[:, -1].values   # Мітки

# Стандартизація даних

scaler = StandardScaler()

X = scaler.fit_transform(X)

# Поділ на навчальну та тестову вибірки

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Перетворення на тензори

X_train = torch.tensor(X_train, dtype=torch.float32)

X_test = torch.tensor(X_test, dtype=torch.float32)

y_train = torch.tensor(y_train, dtype=torch.long)

y_test = torch.tensor(y_test, dtype=torch.long)

# Визначення моделі

class GenomicSwinTransformer(nn.Module):

    def __init__(self):

        super(GenomicSwinTransformer, self).__init__()

        self.swin_transformer = SwinTransformer(img_size=(128, 128), patch_size=4, in_chans=1, num_classes=2, embed_dim=128,
```

```
depths=[2, 2, 6, 2], num_heads=[4, 8, 16, 32], window_size=7, mlp_ratio=4.0, qkv_bias=True, qk_scale=None, drop_rate=0.0, attn_drop_rate=0.0, drop_path_rate=0.2, norm_layer=nn.LayerNorm)


    def forward(self, x):
        x = self.swin_transformer(x)
        return x

model = GenomicSwinTransformer()

# Визначення оптимізатора та функції втрат
optimizer = optim.Adam(model.parameters(), lr=1e-3)
criterion = nn.CrossEntropyLoss()

# Навчання моделі
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    optimizer.zero_grad()


    # Forward pass
    inputs = X_train.unsqueeze(1).to(device)
    labels = y_train.to(device)
    outputs = model(inputs)
    loss = criterion(outputs, labels)
```

```
    # Backward pass і оптимізація
    loss.backward()
    optimizer.step()


    running_loss += loss.item()
    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {running_loss:.4f}')

# Оцінка моделі
model.eval()
with torch.no_grad():
    inputs = X_test.unsqueeze(1).to(device)
    labels = y_test.to(device)
    outputs = model(inputs)
    _, predicted = torch.max(outputs, 1)
    accuracy = accuracy_score(labels.cpu(), predicted.cpu())
    print(f'Accuracy: {accuracy:.4f}')
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	74 с.
	Інші документи	
2.	Презентаційні матеріали	22 плакати

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Панченко Т.І.			Аналіз алгоритмів розпізнавання образів нейронною мережею
Перевірив		Носова Я.В.			
Н.контр.		Малєєва І.А.			
Затв.		Філатов В.О.			

Шифр групи	Код напр./спец.
СШІм-22-3	122
ХНУРЕ кафедра ШІ	