```
import os
import numpy as np
import copy

import torch
import torchvision
from torch import nn
from torch.autograd import Variable
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from torchvision import transforms
from torchvision.utils import save_image
import torch.nn.functional as F
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
import torchvision.models as models



def train(train_loader):
    model.train()
    running_loss = 0
    for batch_idx, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        print(batch_idx,end='')
        optimizer.zero_grad()
        outputs = model(images)

        loss = criterion(outputs, labels)
        running_loss += loss.item()

        loss.backward()
        optimizer.step()

    train_loss = running_loss / len(train_loader)
```

```
    return train_loss


def valid(test_loader):
    model.eval()
    running_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (images, labels) in enumerate(test_loader):
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)

            loss = criterion(outputs, labels)
            running_loss += loss.item()

            predicted = outputs.max(1, keepdim=True)[1]
            correct += predicted.eq(labels.view_as(predicted)).sum().item()
            total += labels.size(0)

    val_loss = running_loss / len(test_loader)
    val_acc = correct / total

    return val_loss, val_acc


if __name__ == "__main__":

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(device)
    np.random.seed(1234)
    torch.manual_seed(1234)

    num_epochs = 50
    batch_size = 128
```

```
    learning_rate =0.00025

    for fold in range(3):
        resnet = models.resnet50(pretrained=False)
        resnet.fc = nn.Linear(2048, 2)
        model = resnet.to(device)
        criterion = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

        train_file_dir=str(fold)+'_train_dir¥¥'
        validation_file_dir=str(fld)+'_validation_dir¥¥'
        logging_dir=str(fld)+'_logging_dir¥¥'

        transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor()
        ])
        train_dataset = datasets.ImageFolder(train_file_dir, transform)
        train_data_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

        validation_dataset = datasets.ImageFolder(validation_file_dir, transform)
        validation_data_loader = DataLoader(validation_dataset, batch_size=batch_size,
shuffle=False)

        best_val_acc=0

        for epoch in range(num_epochs):
            loss = train(train_data_loader)
            val_loss, val_acc = valid(validation_data_loader)
            print('epoch %d, loss: %.4f val_loss: %.4f val_acc: %.4f' % (epoch, loss, val_loss,
val_acc))

            if val_acc > best_val_acc :
                best_val_acc = val_acc
                print("Saving best model")
                best_model_wts = copy.deepcopy(model.state_dict())
```

```
print("Best validation accuracy: {:4f}".format(best_val_acc))
torch.save(best_model_wts, os.path.join(logging_dir, "model_resnet50.pkl"))
```