Бозоров Руслан Миралиевич, группа 20-1

Лабораторная работа № 2

**Цель работы**

Выбрать и решить задачу классификации изображений. Выбрать на ваше усмотрение необходимое количество классов для вашей задачи. Точность вашей модели на тестовых данных не должна быть ниже 0.75.

**Задание**

1. Решить задачу классификации исходного изображения с помощью глубокой сверточной нейронной сети (арифметические операции).
2. Оценить точность полученной модели.
3. Не использовать передобученную нейронную сеть.

**Код программы / ответ на вопрос и т.п.**

import numpy as np

import torch

import torch.nn as nn

import torchvision

from torchvision import datasets

from torchvision import transforms

from torch.optim import Adam

from tqdm import tqdm

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

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

# Создание и нормализация обучающей, тестовой и выборки валидации

batch\_size = 128

transform = transforms.Compose([transforms.ToTensor(),

                                transforms.Resize((128, 128)),

                                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train\_path = "C:/Users/das-s/Downloads/Plants Classification/train"

train\_ds = datasets.ImageFolder(root=train\_path, transform=transform)

train\_dataloader = DataLoader(train\_ds, batch\_size=batch\_size, shuffle=True, pin\_memory=True)

test\_path = "C:/Users/das-s/Downloads/Plants Classification/test"

test\_ds = datasets.ImageFolder(root=test\_path, transform=transform)

test\_dataloader = DataLoader(test\_ds, batch\_size=batch\_size, shuffle=True, pin\_memory=True)

validation\_path = "C:/Users/das-s/Downloads/Plants Classification/val"

validation\_ds = datasets.ImageFolder(root=validation\_path, transform=transform)

validation\_dataloader = DataLoader(validation\_ds, batch\_size=batch\_size, shuffle=True, pin\_memory=True)

# Визуализация изображений и описаний

def imshow(img):

    img = img / 2 + 0.5

    npimg = img.numpy()

    plt.imshow(np.transpose(npimg, (1, 2, 0)))

    plt.show()

train\_features, train\_labels = next(iter(train\_dataloader))

print(f"Feature batch shape: {train\_features.size()}")

print(f"Labels batch shape: {train\_labels.size()}")

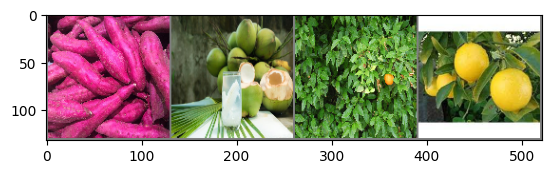
images = []

for i in range(4):

    images.append(train\_features[i])

imshow(torchvision.utils.make\_grid(images)) # show images

print(' '.join(f'{train\_ds.classes[train\_labels[j]]}\t' for j in range(4))) # print labels

****

**sweetpotatoes coconut orange orange**

# Модель свёрточной нейросети

class CNN(nn.Module):

    def \_\_init\_\_(self, channel, num\_classes):

        super(CNN, self).\_\_init\_\_()

        self.layer1 = nn.Sequential(nn.Conv2d(channel, 32, kernel\_size=3),

                                    nn.BatchNorm2d(32),

                                    nn.ReLU(),

                                    nn.MaxPool2d(kernel\_size=2, stride=2))

        self.layer2 = nn.Sequential(nn.Conv2d(32, 64, kernel\_size=3),

                                    nn.BatchNorm2d(64),

                                    nn.ReLU(),

                                    nn.MaxPool2d(kernel\_size=2, stride=2))

        self.layer3 = nn.Sequential(nn.Conv2d(64, 128, kernel\_size=3),

                                    nn.ReLU(),

                                    nn.Conv2d(128, 128, kernel\_size=3),

                                    nn.BatchNorm2d(128),

                                    nn.ReLU(),

                                    nn.MaxPool2d(kernel\_size=2, stride=2))

        self.layer4 = nn.Sequential(nn.Conv2d(128, 256, kernel\_size=3),

                                    nn.ReLU(),

                                    nn.Conv2d(256, 256, kernel\_size=3),

                                    nn.ReLU(),

                                    nn.Conv2d(256, 256, kernel\_size=3),

                                    nn.BatchNorm2d(256),

                                    nn.ReLU(),

                                    nn.MaxPool2d(kernel\_size=2, stride=2))

        self.fc1 = nn.Sequential(

            nn.Dropout(0.3),

            nn.Linear(256\*3\*3, 512),

            nn.ReLU())

        self.fc2 = nn.Sequential(

            nn.Dropout(0.3),

            nn.Linear(512, 128),

            nn.ReLU(),

            nn.Dropout(0.5))

        self.fc3 = nn.Linear(128, num\_classes)

    def forward(self, x):

        x = self.layer1(x)

        x = self.layer2(x)

        x = self.layer3(x)

        x = self.layer4(x)

        x = x.view(-1, 256\*3\*3)

        x = self.fc1(x)

        x = self.fc2(x)

        x = self.fc3(x)

        return nn.functional.softmax(x, dim=1)

**#** Объявление модели

model = CNN(3, len(test\_ds.classes))

model.to(device)

# Функция обучения модели

def train(model, train\_dataloader, optimizer, loss\_func):

    losses = []

    model.train()

    print('Train')

    train\_running\_loss = 0.0

    train\_running\_correct = 0

    counter = 0

    for data, labels in tqdm(train\_dataloader):

        counter += 1

        data = data.to(device)

        labels = labels.to(device)

        outputs = model(data)

        # calculate the loss

        loss = loss\_func(outputs, labels)

        train\_running\_loss += loss.item()

        losses.append(loss.item())

        # calculate the accuracy

        \_, preds = torch.max(outputs.data, 1)

        train\_running\_correct += (preds == labels).sum().item()

        loss.backward()

        optimizer.step()

        optimizer.zero\_grad()

    # loss and accuracy for the complete epoch

    epoch\_loss = train\_running\_loss / counter

    epoch\_acc = 100. \* (train\_running\_correct / len(train\_dataloader.dataset))

    return epoch\_loss, epoch\_acc, losses

**#** Функция валидации модели

def validate(model, validation\_dataloader, loss\_func):

    model.eval()

    print('Validation')

    valid\_running\_loss = 0.0

    valid\_running\_correct = 0

    counter = 0

    with torch.no\_grad():

        for data, labels in tqdm(validation\_dataloader):

            counter += 1

            data = data.to(device)

            labels = labels.to(device)

            # forward pass

            outputs = model(data)

            # calculate the loss

            loss = loss\_func(outputs, labels)

            valid\_running\_loss += loss.item()

            # calculate the accuracy

            \_, preds = torch.max(outputs.data, 1)

            valid\_running\_correct += (preds == labels).sum().item()

    # loss and accuracy for the complete epoch

    epoch\_loss = valid\_running\_loss / counter

    epoch\_acc = 100. \* (valid\_running\_correct / len(validation\_dataloader.dataset))

    return epoch\_loss, epoch\_acc

# Объявление параметров обучения нейросети

optimizer = Adam(params=model.parameters(), lr=1e-3)

loss\_func = nn.CrossEntropyLoss()

num\_epochs = 20

# Процесс обучения нейросети

train\_loss, valid\_loss = [], []

train\_acc, valid\_acc = [], []

all\_t\_losses = []

for epoch in range(num\_epochs):

    print(f"Epoch {epoch+1} of {num\_epochs}")

    train\_epoch\_loss, train\_epoch\_acc, all\_t\_epoch\_losses = train(model, train\_dataloader,

                                              optimizer, loss\_func)

    valid\_epoch\_loss, valid\_epoch\_acc = validate(model, validation\_dataloader,

                                                 loss\_func)

    train\_loss.append(train\_epoch\_loss)

    valid\_loss.append(valid\_epoch\_loss)

    train\_acc.append(train\_epoch\_acc)

    valid\_acc.append(valid\_epoch\_acc)

    all\_t\_losses.append(all\_t\_epoch\_losses)

    print(f"Training loss: {train\_epoch\_loss:.3f}, training acc: {train\_epoch\_acc:.3f}")

    print(f"Validation loss: {valid\_epoch\_loss:.3f}, validation acc: {valid\_epoch\_acc:.3f}")

    print('-'\*50)

Epoch 1 of 30

Train

100%|██████████| 49/49 [00:20<00:00, 2.44it/s]

Validation

100%|██████████| 7/7 [00:02<00:00, 2.66it/s]

Training loss: 1.979, training acc: 38.878

Validation loss: 2.043, validation acc: 32.333

--------------------------------------------------

Epoch 2 of 30

Train

100%|██████████| 49/49 [00:19<00:00, 2.46it/s]

Validation

100%|██████████| 7/7 [00:02<00:00, 2.79it/s]

Training loss: 1.913, training acc: 45.218

Validation loss: 1.924, validation acc: 43.778

--------------------------------------------------

...

--------------------------------------------------

Train

100%|██████████| 49/49 [00:19<00:00, 2.46it/s]

Validation

100%|██████████| 7/7 [00:02<00:00, 2.67it/s]Training loss: 1.765, training acc: 60.423

Validation loss: 1.818, validation acc: 55.000

--------------------------------------------------

# Визуализация точности модели по эпохам

plt.figure(figsize=(10, 7))

plt.plot(

    train\_acc, color='green', linestyle='-',

    label='train accuracy'

)

plt.plot(

    valid\_acc, color='blue', linestyle='-',

    label='validataion accuracy'

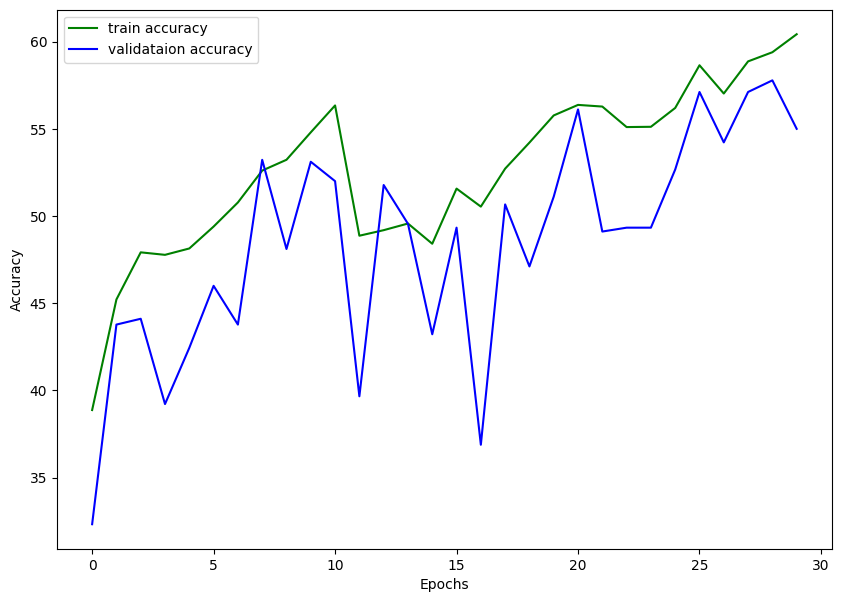
)

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()



# Визуализация ошибок модели по эпохам

plt.figure(figsize=(10, 7))

plt.plot(

    train\_loss, color='orange', linestyle='-',

    label='train loss')

plt.plot(

    valid\_loss, color='red', linestyle='-',

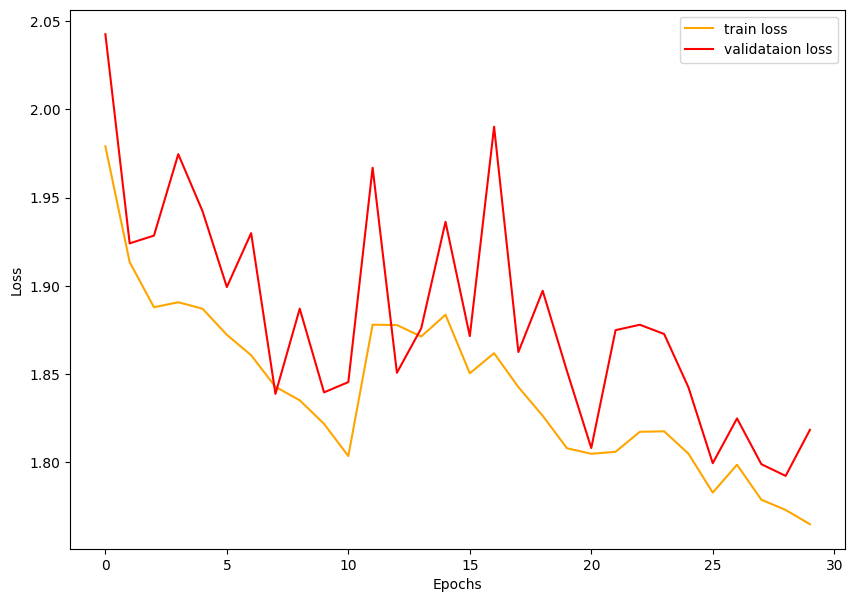
    label='validataion loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()



**Результаты выполнения**

# Тест модели

test\_fetures, test\_labels = next(iter(test\_dataloader))

images = []

for i in range(4):

    images.append(test\_fetures[i])

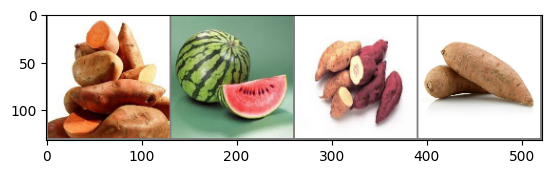
imshow(torchvision.utils.make\_grid(images)) # show images

print("Real value:")

print(' '.join(f'{train\_ds.classes[test\_labels[j]]}\t' for j in range(4))) # print labels

print("Predicted value:")

print(' '.join(f'{train\_ds.classes[model(test\_fetures[j].view(1, 3, 75, 75).to(device)).argmax()]}\t' for j in range(4)))

****

Real value:

sweetpotatoes watermelon sweetpotatoes sweetpotatoes

Predicted value:

sweetpotatoes watermelon sweetpotatoes sweetpotatoes

# Подсчёт точности модели

real\_labels = []

predicted\_labels = []

for data, labels in test\_dataloader:

    data = data.to(device)

    output = model(data).argmax(dim=1).cpu().detach().numpy()

    for i in range(len(output)):

        predicted\_labels.append(output[i])

        real\_labels.append(labels[i].detach().item())

from sklearn.metrics import classification\_report

print(classification\_report(predicted\_labels, real\_labels, target\_names=test\_ds.classes))

precision recall f1-score support

aloevera 0.43 0.74 0.54 117

banana 0.00 0.00 0.00 0

coconut 0.46 0.46 0.46 196

cucumber 0.69 0.46 0.55 298

orange 0.34 0.37 0.35 185

papaya 0.81 0.71 0.76 227

spinach 0.52 0.40 0.45 255

sweetpotatoes 0.80 0.59 0.68 272

watermelon 0.94 0.78 0.85 242

accuracy 0.55 1792

macro avg 0.55 0.50 0.52 1792

weighted avg 0.65 0.55 0.59 1792

**Выводы**

В ходе работы была создана и обучена свёрточная нейронная сеть для классификации изображений. Точность модели