Хаустов Игорь Сергеевич, 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.**

# Подключение необходимых библиотек

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

# Переключение работы нейросети на GPU, если это возможно

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 = "Datasets/Vehicle/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 = "Datasets/Vehicle/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 = "Datasets/Vehicle/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     # unnormalize

    npimg = img.numpy()

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

    plt.show()

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

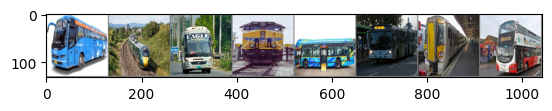
images = []

for i in range(8):

    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(8))) # print labels

****

**Buses Trains Buses Trains Buses Buses Trains Buses**

# Модель свёрточной нейросети

class CNN(nn.Module):

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

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

        self.layer1 = nn.Sequential(nn.Conv2d(channel, 10, kernel\_size=3),

                                    nn.BatchNorm2d(10),

                                    nn.ReLU(),

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

        self.layer2 = nn.Sequential(nn.Conv2d(10, 20, kernel\_size=3),

                                    nn.BatchNorm2d(20),

                                    nn.ReLU(),

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

        self.layer3 = nn.Sequential(nn.Conv2d(20, 30, kernel\_size=3),

                                    nn.BatchNorm2d(30),

                                    nn.ReLU(),

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

        self.layer4 = nn.Sequential(nn.Conv2d(30, 40, kernel\_size=3),

                                    nn.BatchNorm2d(40),

                                    nn.ReLU())

        self.fc1 = nn.Sequential(

            nn.Dropout(0.4),

            nn.Linear(40\*12\*12, 512),

            nn.ReLU())

        self.fc2 = nn.Sequential(

            nn.Dropout(0.4),

            nn.Linear(512, 128),

            nn.ReLU())

        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, 40\*12\*12)

        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 = 40

# 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 50

Train

100%|██████████| 19/19 [00:41<00:00, 2.20s/it]

Validation

100%|██████████| 13/13 [00:33<00:00, 2.60s/it]

Training loss: 1.994, training acc: 25.855

Validation loss: 2.086, validation acc: 15.125

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

Epoch 2 of 50

Train

100%|██████████| 19/19 [00:43<00:00, 2.27s/it]

Validation

100%|██████████| 13/13 [00:33<00:00, 2.59s/it]

Training loss: 1.858, training acc: 41.910

Validation loss: 1.897, validation acc: 36.875

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

...

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

Epoch 50 of 50

Train

100%|██████████| 19/19 [00:43<00:00, 2.28s/it]

Validation

100%|██████████| 13/13 [00:33<00:00, 2.60s/it]

Training loss: 1.318, training acc: 95.746

Validation loss: 1.559, validation acc: 71.125

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

# 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'

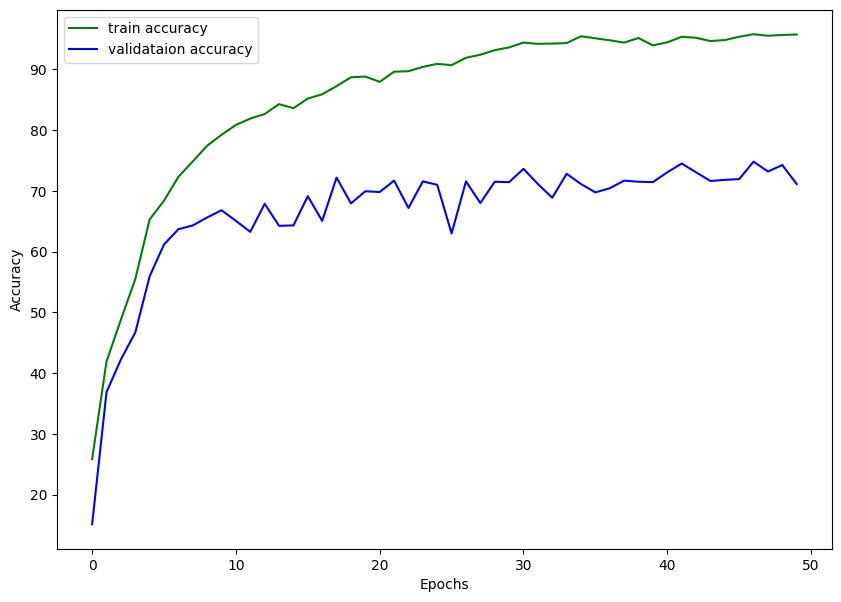
)

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='-',

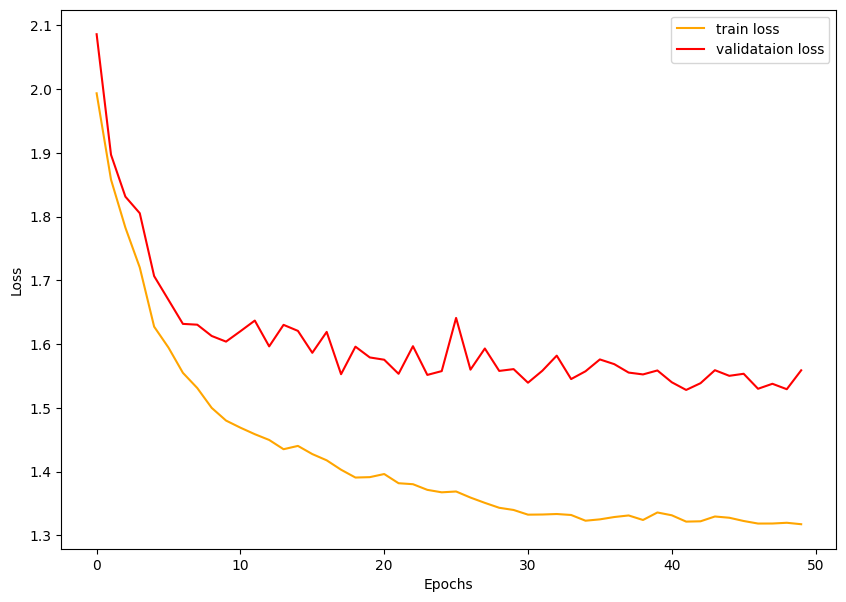
    label='validataion loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()



# Видно, что графики не расходятся, значит модель не переобучилась

**Результаты выполнения**

# Тест модели

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

images = []

for i in range(4):

    images.append(test\_fetures[i])

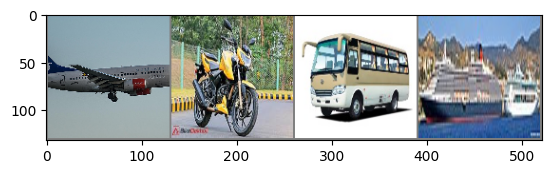
imshow(torchvision.utils.make\_grid(images)) # show images

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:**

**Airplanes Bikes Buses Ships**

**Predicted value:**

**Airplanes Bikes Buses Ships**

# 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

Airplanes 0.89 0.61 0.72 145

Bikes 0.88 0.89 0.88 98

Buses 0.64 0.29 0.40 48

Cars 0.80 0.79 0.79 100

Helicopters 0.68 0.61 0.65 83

Ships 0.73 0.95 0.82 76

Trains 0.62 0.68 0.65 82

Trucks 0.43 0.95 0.59 39

accuracy 0.72 671

macro avg 0.71 0.72 0.69 671

weighted avg 0.75 0.72 0.72 671

**Выводы**

В результате выполнения лабораторной работы была создана и обучена глубокая свёрточная нейронная сеть для классификации изображений различных типов транспортных средств. Для этого была выбрана библиотека pytorch, которая предоставляет удобные инструменты для работы с нейронными сетями. После обучения модели, была оценена ее точность. Полученный результат составил 72%, что свидетельствует о достаточно высоком качестве модели.