Кулиев Фёдор Русланович, 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.**

Для решения данной задачи был выбран набор данных с различными эмоциями людей.

# 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 = "C:/Users/das-s/Downloads/Faces/train"

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

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

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

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

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

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

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

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

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(8))) # print labels

****

**Neutral Happy Happy Angry**

# 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())

        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 = 30

# 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%|██████████| 111/111 [01:07<00:00, 1.65it/s]

Validation

100%|██████████| 7/7 [00:04<00:00, 1.75it/s]

Training loss: 1.468, validation loss: 1.552

Training acc: 41.297, validation acc: 30.714

Epoch 2 of 30

Train

100%|██████████| 111/111 [01:03<00:00, 1.74it/s]

Validation

100%|██████████| 7/7 [00:03<00:00, 1.94it/s]

Training loss: 1.373, validation loss: 1.522

Training acc: 52.077, validation acc: 33.265

.

.

.

Epoch 30 of 30

Train

100%|██████████| 111/111 [01:02<00:00, 1.78it/s]

Validation

100%|██████████| 7/7 [00:03<00:00, 1.97it/s]

Training loss: 1.132, validation loss: 1.209

Training acc: 76.902, validation acc: 63.571

# График 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'

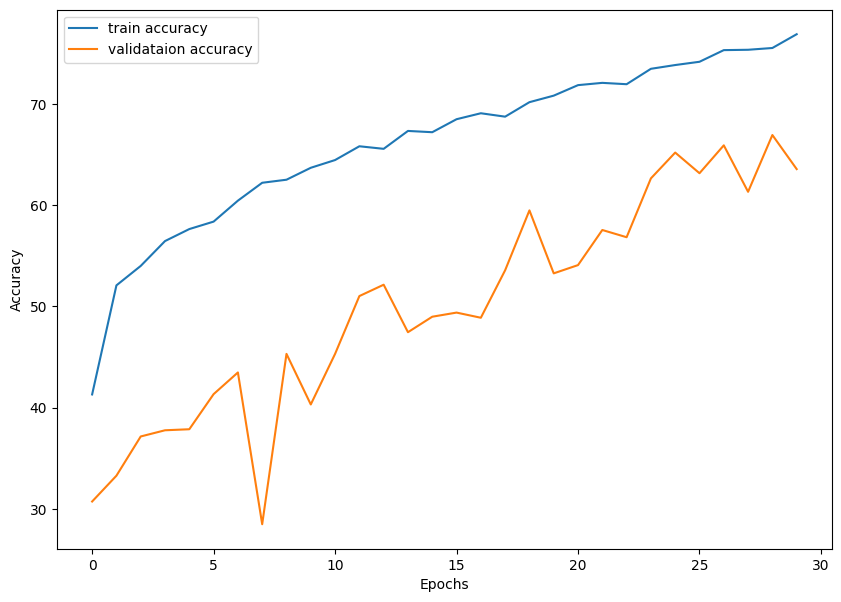
)

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()



# График 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='-',

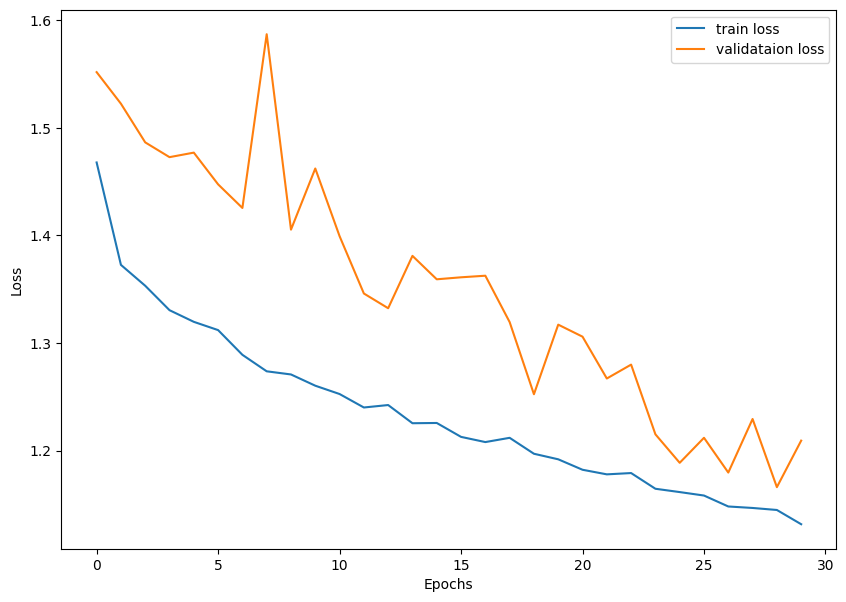
    label='validataion loss')

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:**

**Neutral Sad Happy Surprise**

**Predicted value:**

**Neutral Sad Happy Surprise**

# 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

Angry 0.42 0.87 0.57 269

Happy 0.92 0.79 0.85 725

Neutral 0.72 0.56 0.63 832

Sad 0.76 0.68 0.72 915

Surprise 0.72 0.86 0.79 459

accuracy 0.72 3200

macro avg 0.71 0.75 0.71 3200

weighted avg 0.75 0.72 0.72 3200

**Выводы**

В результате выполнения лабораторной работы была создана и обучена глубокая свёрточная нейронная сеть для классификации изображений различных эмоций человека. После обучения модели, была оценена ее точность. Полученный результат составил 72%, что свидетельствует о достаточно высоком качестве модели.