Кимбрис Владимир Владимирович, 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 = 512

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

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

train\_path = "C:/Users/Vova/Downloads/Cards/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/Vova/Downloads/Cards/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/Vova/Downloads/Cards/valid"

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))

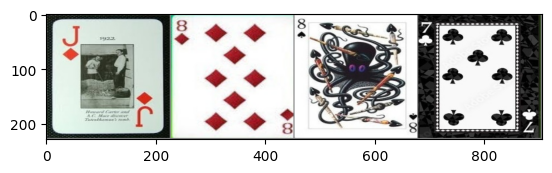
images = []

for i in range(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

****

**jack of diamonds eight of diamonds eight of spades seven of clubs**

# Модель свёрточной нейросети

class CNN(nn.Module):

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

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

        self.layer1 = nn.Sequential(nn.Dropout(0.3),

                                    nn.Conv2d(channel, 16, kernel\_size=3),

                                    nn.BatchNorm2d(16),

                                    nn.ReLU(),

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

        self.layer2 = nn.Sequential(nn.Dropout(0.3),

                                    nn.Conv2d(16, 32, kernel\_size=3),

                                    nn.BatchNorm2d(32),

                                    nn.ReLU(),

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

        self.layer3 = nn.Sequential(nn.Dropout(0.3),

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

                                    nn.BatchNorm2d(64),

                                    nn.ReLU(),

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

        self.layer4 = nn.Sequential(nn.Dropout(0.3),

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

                                    nn.BatchNorm2d(128),

                                    nn.ReLU(),

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

        self.layer5 = nn.Sequential(nn.Dropout(0.3),

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

                                    nn.BatchNorm2d(256),

                                    nn.ReLU())

        self.fc1 = nn.Sequential(

            nn.Flatten(),

            nn.Dropout(0.4),

            nn.Linear(256\*10\*10, 1024),

            nn.ReLU())

        self.fc2 = nn.Sequential(

            nn.Dropout(0.4),

            nn.Linear(1024, num\_classes))

    def forward(self, x):

        x = self.layer1(x)

        x = self.layer2(x)

        x = self.layer3(x)

        x = self.layer4(x)

        x = self.layer5(x)

        x = self.fc1(x)

        x = self.fc2(x)

        return nn.functional.log\_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 40

Train

100%|██████████| 11/11 [00:16<00:00, 1.53s/it]

Validation

100%|██████████| 1/1 [00:00<00:00, 3.87it/s]

Training loss: 6.485, training acc: 5.024

Validation loss: 3.174, validation acc: 12.222

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

Epoch 2 of 40

Train

100%|██████████| 11/11 [00:17<00:00, 1.62s/it]

Validation

100%|██████████| 1/1 [00:00<00:00, 3.42it/s]

Training loss: 3.180, training acc: 15.090

Validation loss: 2.780, validation acc: 19.444

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

...

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

Epoch 40 of 40

Train

100%|██████████| 11/11 [00:19<00:00, 1.75s/it]

Validation

100%|██████████| 1/1 [00:00<00:00, 3.04it/s]

Training loss: 0.705, training acc: 77.298

Validation loss: 1.013, validation acc: 75.556

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

# 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'

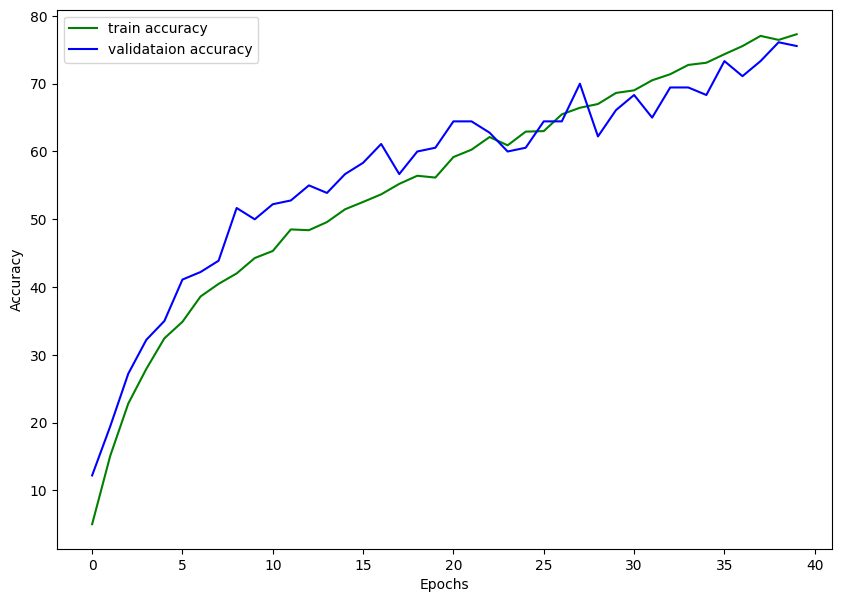
)

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='-',

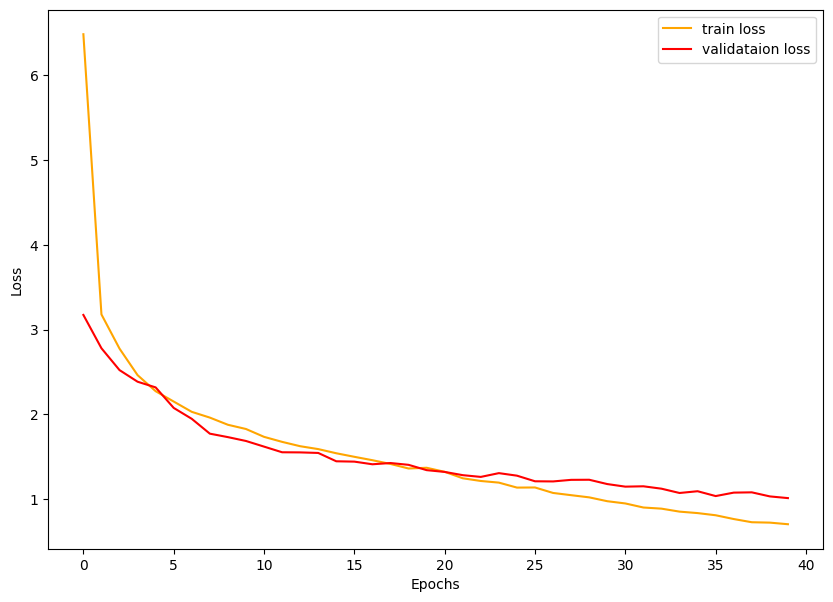
    label='validataion loss')

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])

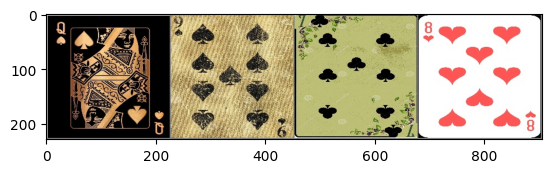
imshow(torchvision.utils.make\_grid(images)) # show images

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:**

**queen of spades nine of spades seven of clubs eight of hearts**

**Predicted value:**

**queen of spades nine of spades seven of spades eight of hearts**

**# 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**

**ace of clubs 0.80 0.67 0.73 6**

**ace of diamonds 0.80 0.80 0.80 5**

**ace of hearts 1.00 0.56 0.71 9**

**ace of spades 0.80 0.50 0.62 8**

**eight of clubs 0.80 1.00 0.89 4**

**eight of diamonds 1.00 1.00 1.00 5**

**eight of hearts 0.80 0.67 0.73 6**

**eight of spades 1.00 1.00 1.00 5**

**jack of clubs 0.60 1.00 0.75 3**

**jack of diamonds 0.60 0.75 0.67 4**

**jack of hearts 0.60 1.00 0.75 3**

**jack of spades 0.60 0.60 0.60 5**

**king of clubs 0.80 0.67 0.73 6**

**king of diamonds 1.00 0.62 0.77 8**

**king of hearts 0.80 0.80 0.80 5**

**king of spades 0.40 0.67 0.50 3**

**nine of clubs 0.20 0.50 0.29 2**

**nine of diamonds 0.80 0.67 0.73 6**

**nine of hearts 0.80 1.00 0.89 4**

**nine of spades 0.60 0.60 0.60 5**

**queen of clubs 0.80 0.50 0.62 8**

**queen of diamonds 0.60 0.75 0.67 4**

**queen of hearts 0.60 0.60 0.60 5**

**queen of spades 0.80 0.67 0.73 6**

**seven of clubs 0.80 1.00 0.89 4**

**seven of diamonds 1.00 1.00 1.00 5**

**seven of hearts 1.00 1.00 1.00 5**

**seven of spades 0.80 0.67 0.73 6**

**six of clubs 0.40 0.50 0.44 4**

**six of diamonds 1.00 0.83 0.91 6**

**six of hearts 0.60 1.00 0.75 3**

**six of spades 0.80 0.67 0.73 6**

**ten of clubs 0.60 1.00 0.75 3**

**ten of diamonds 0.60 1.00 0.75 3**

**ten of hearts 0.80 0.80 0.80 5**

**ten of spades 0.80 0.80 0.80 5**

**accuracy 0.74 180**

**macro avg 0.74 0.77 0.74 180**

**weighted avg 0.78 0.74 0.75 180**

**Выводы**

В результате выполнения работы, я создал и обучил модель глубокой свёрточной нейронной сети, которая способна решать задачу классификации игральных карт. Для реализации данной задачи была использована библиотека pytorch. Точность полученной модели 74%.