Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Домашняя работа №5

по курсу «Теория искусственных нейронных сетей»

«Сверточные нейронные сети (CNN)»

Студент группы ИУ9-71Б Окутин Д. А.

Преподаватель Каганов Ю. Т.

Москва 2024

1 Цель

Ознакомление с основными архитектурами свёрточных нейронных сетей.

2 Задание

1. Реализовать нейронную сеть архитектуры LeNet и проверить её на датасете MNIST.

2. Реализовать нейронную сеть архитектуры VGG16 и проверить её на датасете CIFAR10.

3. Реализовать нейронную сеть архитектуры ResNet и проверить её на датасете CIFAR10.

3 Реализация

Исходный код представлен в листинге 1 - **??**.

Листинг 1: Подготовка датасета

```

 from torchvision.datasets import MNIST, cifar
from torch.utils.data import DataLoader
from torchvision import transforms

from matplotlib import pyplot as plt
import numpy as np
from IPython.display import clear_output
import torch
from torch import nn
import sys
from tqdm import tqdm

train_dataset = MNIST('.', train=True, download=True,transform=
    transforms.ToTensor())
test_dataset = MNIST('.', train=False,transform=transforms.ToTensor())

train_loader_MNIST = DataLoader(train_dataset, batch_size=32, shuffle=
    True)
test_loader_MNIST = DataLoader(test_dataset, batch_size=32, shuffle=
    False)

```

```
t = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994,
    0.2010)),
])

train_dataset = cifar.CIFAR10(root='data',train=True,download=True,
    transform=t)
test_dataset = cifar.CIFAR10(root='data',train=False,transform=t)

train_loader_CIFAR = DataLoader(train_dataset, batch_size=32, shuffle=
    True)
test_loader_CIFAR = DataLoader(test_dataset, batch_size=32, shuffle=
    False)

device = 'cuda' if torch.cuda.is_available() else 'cpu'
device
```

Листинг 2: Функция тренировки

```

  def train(network,train_loader, test_loader, epochs,loss_fn,optim,
    plot=True, verbose=True):
    train_loss_epochs = []
    test_loss_epochs = []
    train_accuracy_epochs = []
    test_accuracy_epochs = []

    try:
        for epoch in tqdm(range(epochs)):
            losses = []
            accuracies = []
            for X, y in train_loader:
                X,y = X.to(device), y.to(device)
                pred = model(X)
                loss_batch = loss_fn(pred, y)
                losses.append(loss_batch.item())
                optim.zero_grad()
                loss_batch.backward()
                optim.step()
                accuracies.append((pred.argmax(dim=1) == y).float().mean
    ().item())
            train_loss_epochs.append(np.mean(losses))
            train_accuracy_epochs.append(np.mean(accuracies))

```

```
            with torch.no_grad():
                losses = []
                accuracies = []
                for X, y in test_loader:
                    X,y = X.to(device), y.to(device)
                    pred = model(X)
                    loss_batch = loss_fn(pred, y)
                    losses.append(loss_batch.cpu())
                    accuracies.append((pred.argmax(dim=1) == y).float().mean().item())
            test_loss_epochs.append(np.mean(losses))
            test_accuracy_epochs.append(np.mean(accuracies))
            clear_output(True)
            if verbose:
                sys.stdout.write('\rEpoch {0}... (Train/Test) Loss: {1:.3f}/{2:.3f}\tAccuracy: {3:.3f}/{4:.3f}'.format(
                            epoch, train_loss_epochs[-1], test_loss_epochs[-1],
                            train_accuracy_epochs[-1], test_accuracy_epochs[-1]))
            if plot:
                plt.figure(figsize=(12, 5))
                plt.subplot(1, 2, 1)
                plt.plot(train_loss_epochs, label='Train')
                plt.plot(test_loss_epochs, label='Test')
                plt.xlabel('Epochs', fontsize=16)
                plt.ylabel('Loss', fontsize=16)
                plt.legend(loc=0, fontsize=16)
                plt.grid('on')
                plt.subplot(1, 2, 2)
                plt.plot(train_accuracy_epochs, label='Train accuracy')
                plt.plot(test_accuracy_epochs, label='Test accuracy')
                plt.xlabel('Epochs', fontsize=16)
                plt.ylabel('Accuracy', fontsize=16)
                plt.legend(loc=0, fontsize=16)
                plt.grid('on')
                plt.show()
    except KeyboardInterrupt:
        pass
    return train_loss_epochs, \
           test_loss_epochs, \
           train_accuracy_epochs, \
           test_accuracy_epochs
```

Листинг 3: LeNet

```

```

```python
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()

        self.conv = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5,
    padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        self.fc = nn.Sequential(
            nn.Linear(in_features=16 * 5 * 5, out_features=120),
            nn.ReLU(),
            nn.Linear(in_features=120, out_features=84),
            nn.ReLU(),
            nn.Linear(in_features=84, out_features=10)
        )

    def forward(self, img):
        feature = self.conv(img)
        output = self.fc(feature.view(img.shape[0], -1))
        return output
```

Листинг 4: VGG16

```python

def conv_block(in_channels, out_channels):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
        nn.ReLU(),
        nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2)
    )


def fc_block(size_in, size_out):
    layer = nn.Sequential(
        nn.Linear(size_in, size_out),
        nn.BatchNorm1d(size_out),
        nn.ReLU(),
        nn.Dropout(),
    )
```

```
    return layer


class VGG16(nn.Module):
    def __init__(self):
        super(VGG16, self).__init__()

        self.convs = nn.Sequential(
            conv_block(3, 128),
            conv_block(128, 256),
            conv_block(256, 512),
        )

        self.fc = nn.Sequential(
            fc_block(512*4*4, 1024),
            fc_block(1024, 1024),
            fc_block(1024, 10),
        )

    def forward(self, x):
        x = self.convs(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)

        return x
```

Листинг 5: ResNet

```

class ResidualBlock(nn.Module):
    expansion=1
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels, out_channels,kernel_size=3, stride=
    stride, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU())
        self.conv2 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels,kernel_size=3, stride
    =1, padding=1),
            nn.BatchNorm2d(out_channels))
        self.relu = nn.ReLU()
        self.out_channels = out_channels
        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
                self.shortcut = nn.Sequential(
```

```
                    nn.Conv2d(in_channels, self.expansion * out_channels, kernel_size=1, stride=stride, bias=False),
                    nn.BatchNorm2d(self.expansion * out_channels)
                )

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.conv2(out)
        out += self.shortcut(residual)
        out = self.relu(out)
        return out


class ResNet(nn.Module):
    def __init__(self, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU()
        )
        self.in_planes = 16
        self.layer1 = self._make_layer(ResidualBlock, 16, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(ResidualBlock, 32, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(ResidualBlock, 64, num_blocks[2], stride=2)
        self.avgpool = nn.AvgPool2d(7, stride=2)
        self.fc = nn.Sequential(
            nn.BatchNorm1d(64),
            nn.Linear(64, 128),
            nn.ReLU(128),
            nn.BatchNorm1d(128),
            nn.Linear(128, 10),
        )

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion

        return nn.Sequential(*layers)
```

```


    def forward(self, x):
        x = self.conv1(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)

        return x

def resnet32():
    return ResNet([5, 5, 5])
```

4 Результаты

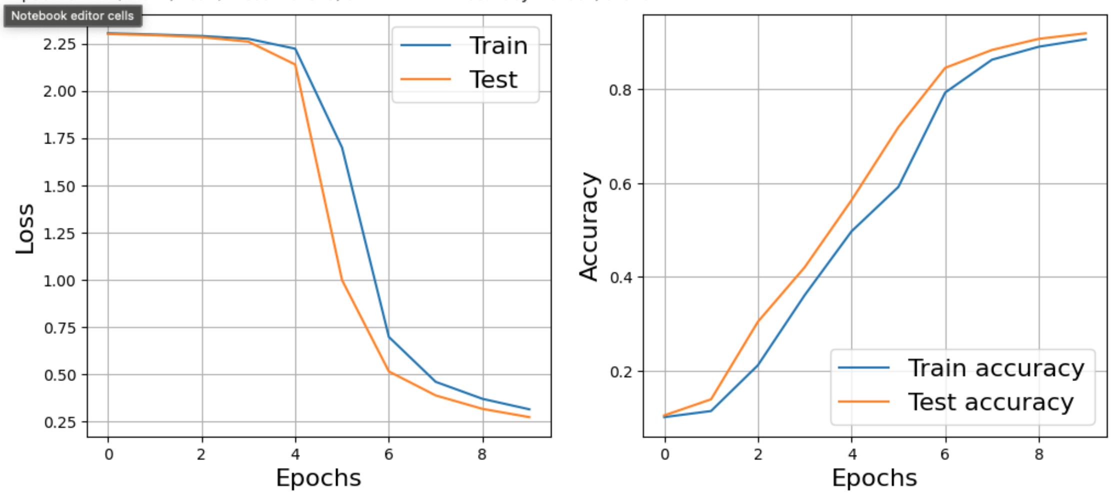
Результат представлен на рисунках 1 - 5.

```
model=LeNet()
model.to(device)
lr = 0.001

loss=nn.CrossEntropyLoss()
optim=torch.optim.SGD(model.parameters(), lr=lr)

tr_lenet_sgd, ts_lenet_sgd, tr_ac_lenet_sgd, ts_ac_lenet_sgd = train(
    model,train_loader_MNIST,test_loader_MNIST, epochs=10, plot=True, verbose=True, loss_fn=loss, optim=optim)
```

```
Epoch 9... (Train/Test) Loss: 0.313/0.272       Accuracy: 0.907/0.920
```




```
100%|██████████| 10/10 [02:01<00:00, 12.20s/it]
```

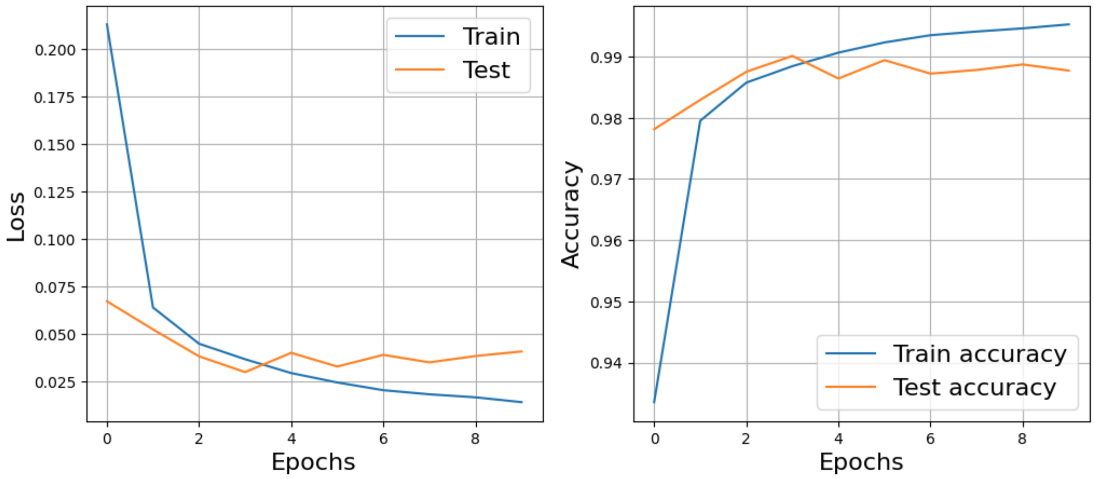
Рис. 1 — LeNet SGD

```
model=LeNet()
model.to(device)
lr = 0.001

loss=nn.CrossEntropyLoss()
optim=torch.optim.Adam(model.parameters(), lr=lr)

tr_lenet_adam, ts_lenet_adam, tr_ac_lenet_adam, ts_ac_lenet_adam = train(
    model,train_loader_MNIST,test_loader_MNIST, epochs=10, plot=True, verbose=True, loss_fn=loss, optim=optim)
```

```
Epoch 9... (Train/Test) Loss: 0.014/0.041       Accuracy: 0.995/0.988
```




```
100%|██████████| 10/10 [02:07<00:00, 12.78s/it]
```

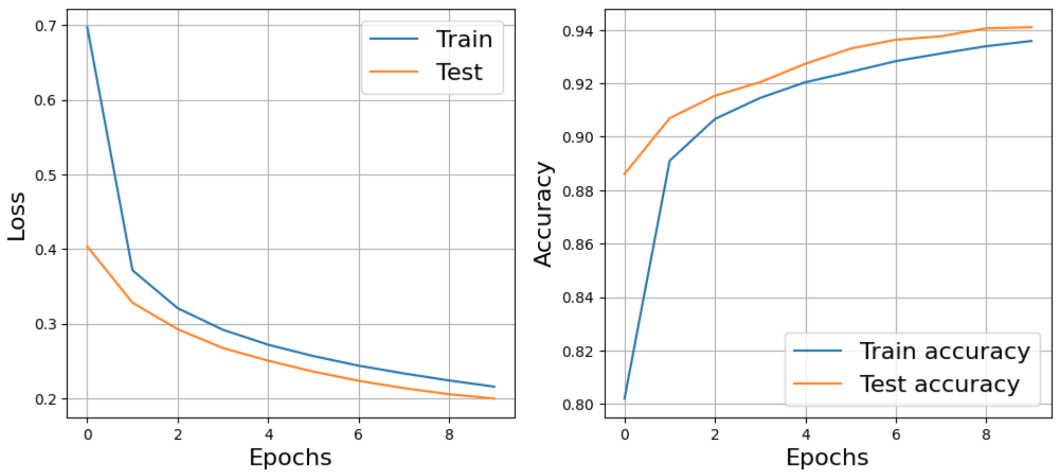
Рис. 2 — LeNet Adam

```
model=LeNet()
model.to(device)
lr = 0.001

loss=nn.CrossEntropyLoss()
optim=torch.optim.Adagrad(model.parameters(), lr=lr)

tr_lenet_adagrad, ts_lenet_adagrad, tr_ac_lenet_adagrad, ts_ac_lenet_adagrad = train(
    model,train_loader_MNIST,test_loader_MNIST, epochs=10, plot=True, verbose=True, loss_fn=loss, optim=optim)
```

```
Epoch 9... (Train/Test) Loss: 0.215/0.200       Accuracy: 0.936/0.941
```




```
100%|██████████| 10/10 [02:07<00:00, 12.75s/it]
```

Рис. 3 — LeNet Adagrad

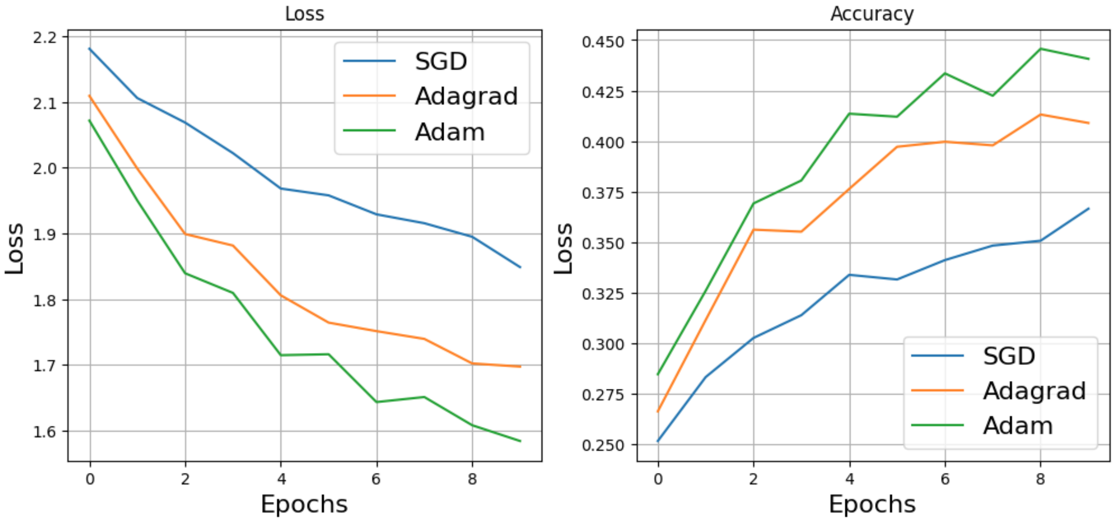


Рис. 4 — VGG16 SGD/Adagrad/Adam

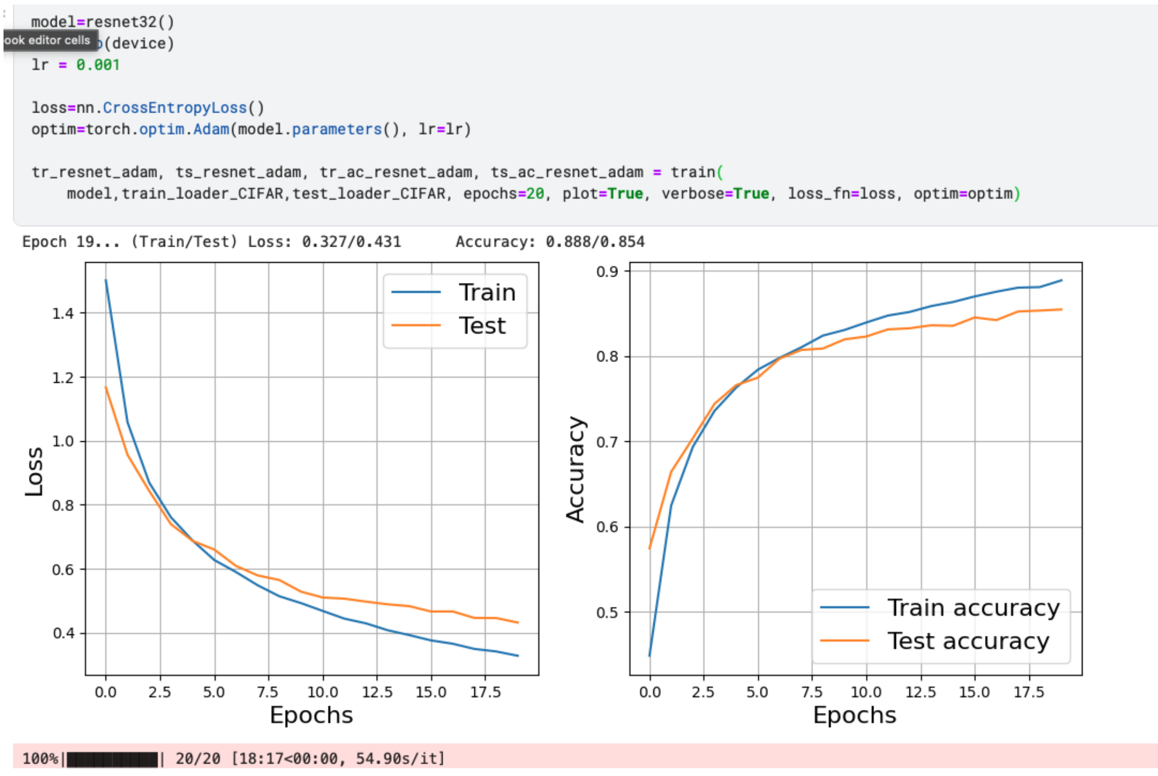


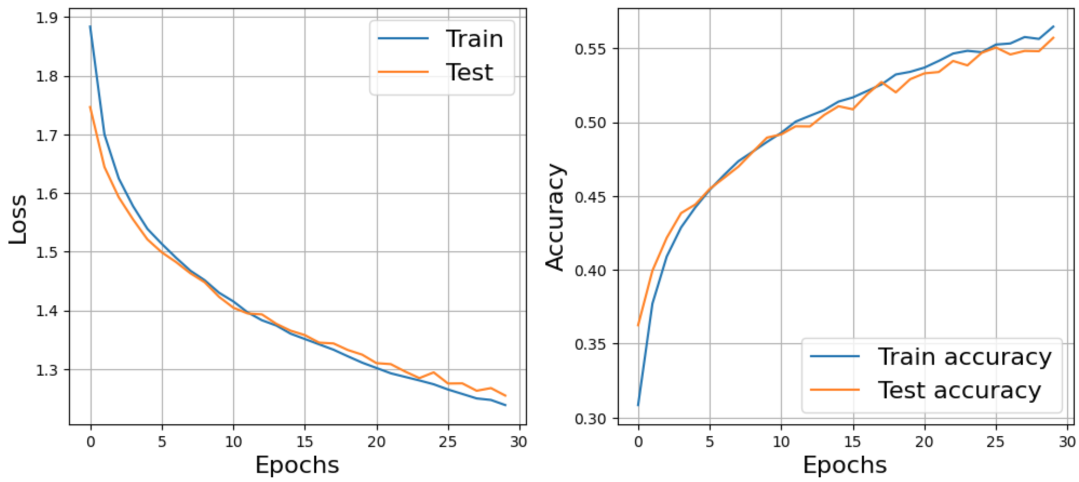
Рис. 5 — ResNet Adam

```
model=resnet32()
model.to(device)
lr = 0.0005

loss=nn.CrossEntropyLoss()
optim=torch.optim.Adagrad(model.parameters(), lr=lr)

tr_resnet_adam, ts_resnet_adam, tr_ac_resnet_adam, ts_ac_resnet_adam = train(
    model,train_loader_CIFAR,test_loader_CIFAR, epochs=30, plot=True, verbose=True, loss_fn=loss, optim=optim)
```

```
Epoch 29... (Train/Test) Loss: 1.239/1.255      Accuracy: 0.565/0.557
```




```
100%|██████████| 30/30 [27:25<00:00, 54.85s/it]
```

Рис. 6 — ResNet Adagrad

5 Выводы

В результе выполнения данной лабораторной работы были реализованы различные архитектуры сверточных нейронных сетей с помощью библиотеки pytorch. Реализованные архитектуры были протестированы на различных открытых датасетах с использованием различных оптимизаторов.