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»

Лабораторная работа № 5

по курсу «Теория искусственных нейронных сетей»

«Сверточные нейронные сети»

Студент группы ИУ9-71Б Баев Д.А

Преподаватель Каганов Ю. Т.

Москва 2023

1 Задание

1. LeNet.

2. VGG.

3. ResNet.

4. Подготовить отчет с распечаткой текста программы, графиками результатов исследования и анализом результатов.

2 Исходный код

Исходный код программы представлен в листингах 1- 5

Листинг 1: Подготовка датасета

```
from torch import nn
from tqdm import tqdm

from torchvision.datasets import mnist, cifar
from torchvision import transforms

from torch.utils.data import DataLoader, Subset
import matplotlib.pyplot as plt
import torch

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

batch_size = 8
batches_per_epoch = 128
test_size_mnist = 128
train_size_mnist = 5120
test_size_cifar = 1024
train_size_cifar = 48976

mnist_dataset = mnist.MNIST(
    root = 'data',
    train=True,
    download=True,
    transform=transforms.ToTensor()
)

mnist_dataloader = {
    "train": DataLoader(Subset(mnist_dataset, range(test_size_mnist,
    test_size_mnist + train_size_mnist)), shuffle=True, batch_size=
    batch_size),
    "test": DataLoader(Subset(mnist_dataset, range(0, test_size_mnist)),
     shuffle=True, batch_size=batch_size)
}

transform = 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))
```

```
])

cifar_dataset = cifar.CIFAR10(
    root='data',
    train=True,
    download=True,
    transform=transform
)

cifar_dataloader = {
    "train": DataLoader(Subset(cifar_dataset, range(test_size_cifar,
    test_size_cifar + train_size_cifar)), shuffle=True, batch_size=
    batch_size),
    "test": DataLoader(Subset(cifar_dataset, range(0, test_size_cifar)),
     shuffle=True, batch_size=batch_size)
}
```

Листинг 2: LeNet

```
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, image):
        output = self.conv(image)
        output = self.fc(output.view(image.shape[0], -1))
        return output
```

Листинг 3: MiniVGG

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

        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        self.classifier = nn.Sequential(
            nn.Linear(128 * 8 * 8, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(512, 256),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(256, 10)
        )

    def forward(self, image):
        output = self.features(image)
        output = output.view(image.shape[0], -1)
        output = self.classifier(output)
        return output
```

Листинг 4: ResNet

```
class ResidualBlock(nn.Module):
    def __init__(self, in_ch, out_ch, stride=1):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_ch, out_ch, kernel_size=3, stride=stride,
    padding=1),
            nn.BatchNorm2d(out_ch),
            nn.ReLU()
        )
        self.conv2 = nn.Sequential(
```

```
10              nn.Conv2d(out_ch, out_ch, kernel_size=3, stride=1, padding
    =1),
11              nn.BatchNorm2d(out_ch)
12          )
13          self.relu = nn.ReLU()
14          if in_ch != out_ch:
15              self.downsaple = nn.Conv2d(in_ch, out_ch, kernel_size=3,
    stride=2, padding=1)
16          else:
17              self.downsaple = None
18
19      def forward(self, image):
20          residual = image
21          output = self.conv1(image)
22          output = self.conv2(output)
23          if self.downsaple is not None:
24              residual = self.downsaple(residual)
25          output += residual
26          output = self.relu(output)
27          return output
28
29  class ResNet(nn.Module):
30      def __init__(self):
31          super(ResNet, self).__init__()
32          self.conv1 = nn.Sequential(
33              nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1),
34              nn.BatchNorm2d(16),
35              nn.ReLU()
36          )
37          self.layer1 = ResidualBlock(16, 16, 1)
38          self.layer2 = ResidualBlock(16, 32, 2)
39          self.layer3 = ResidualBlock(32, 64, 2)
40          self.avgpool = nn.AvgPool2d(7, stride=2)
41          self.fc = nn.Sequential(
42              nn.BatchNorm1d(64),
43              nn.Linear(64, 128),
44              nn.ReLU(),
45              nn.BatchNorm1d(128),
46              nn.Linear(128, 10)
47          )
48      def forward(self, image):
49          output = self.conv1(image)
50          output = self.layer1(output)
51          output = self.layer2(output)
52          output = self.layer3(output)
53          output = self.avgpool(output)
```

```
        output = output.view(image.shape[0], -1)
        output = self.fc(output)
        return output
```

Листинг 5: Обучение модели

```
def train_model_and_plot(model, optimizer, dataloader, epochs=50, lr=0.001):
    model.to(device)
    train_acc, test_acc = [], []
    if optimizer == "NAG":
        optimizer = torch.optim.SGD(model.parameters(), lr=lr, nesterov=True, momentum=0.9)
    else:
        optimizer = optimizer(model.parameters(), lr=lr)
    loss_fn = nn.CrossEntropyLoss()

    for epoch in tqdm(range(epochs)):
        if epoch == 200:
            optimizer.param_groups[0]['lr'] /= 10
        running_accuracy = 0
        i = 0
        for x, y in dataloader["train"]:
            x, y = x.to(device), y.to(device)
            prediction = model(x)
            loss = loss_fn(prediction, y)
            running_accuracy += (prediction.softmax(dim=1).argmax(dim=1) == y).float().mean()
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            i += 1
            if i == batches_per_epoch:
                break
        running_accuracy /= batches_per_epoch
        train_acc.append(running_accuracy.cpu())

        with torch.no_grad():
            running_accuracy = 0
            for x, y in dataloader['test']:
                x, y = x.to(device), y.to(device)
                prediction = model(x)
                running_accuracy += (prediction.softmax(dim=1).argmax(dim=1) == y).float().mean()
            running_accuracy /= len(dataloader['test'])
            test_acc.append(running_accuracy.cpu())
    plt.plot(range(1, epochs + 1), train_acc, label="train")
```

```
38      plt.plot(range(1, epochs + 1), test_acc, label="test")
39      plt.legend()
40      plt.show()
41      print(train_acc[-1])
42      print(test_acc[-1])
```

3 Результаты

На рисунке 1 приведен график точности модели LeNet с оптимизатором SGD и скоростью обучения 0.0055.

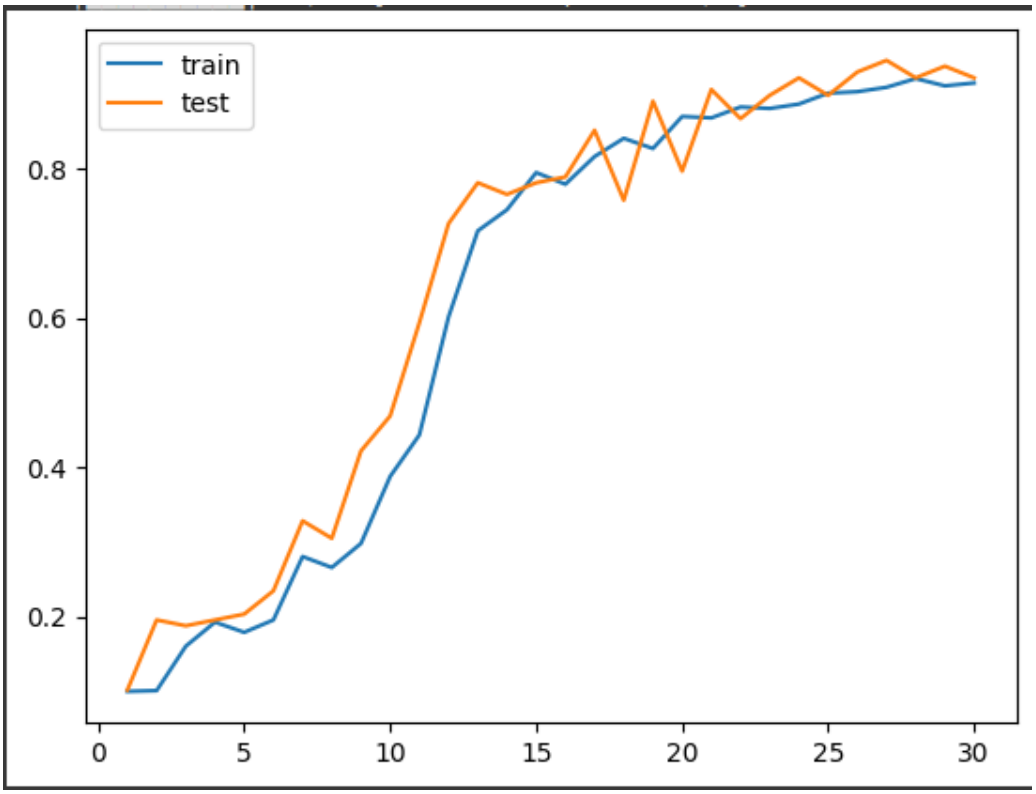

Рис. 1

На рисунке 2 приведен график точности модели LeNet с оптимизатором Adadelta и скоростью обучения 0.1.

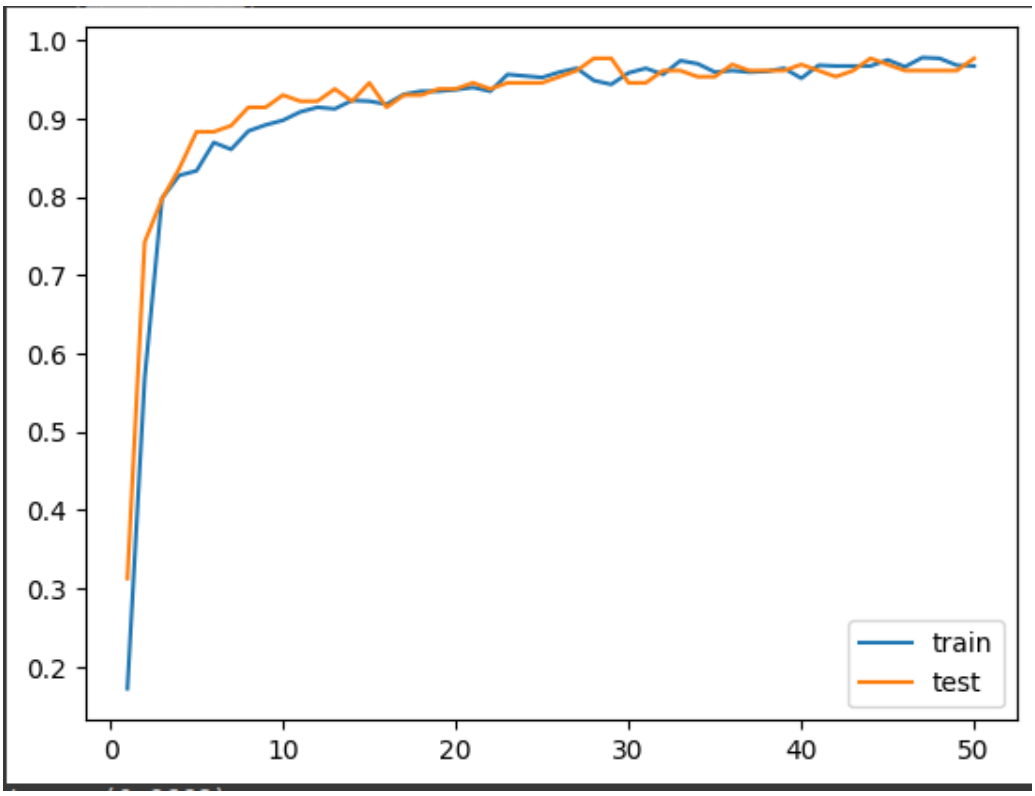

Рис. 2

На рисунке 3 приведен график точности модели LeNet с оптимизатором NAG и скоростью обучения 0.001.

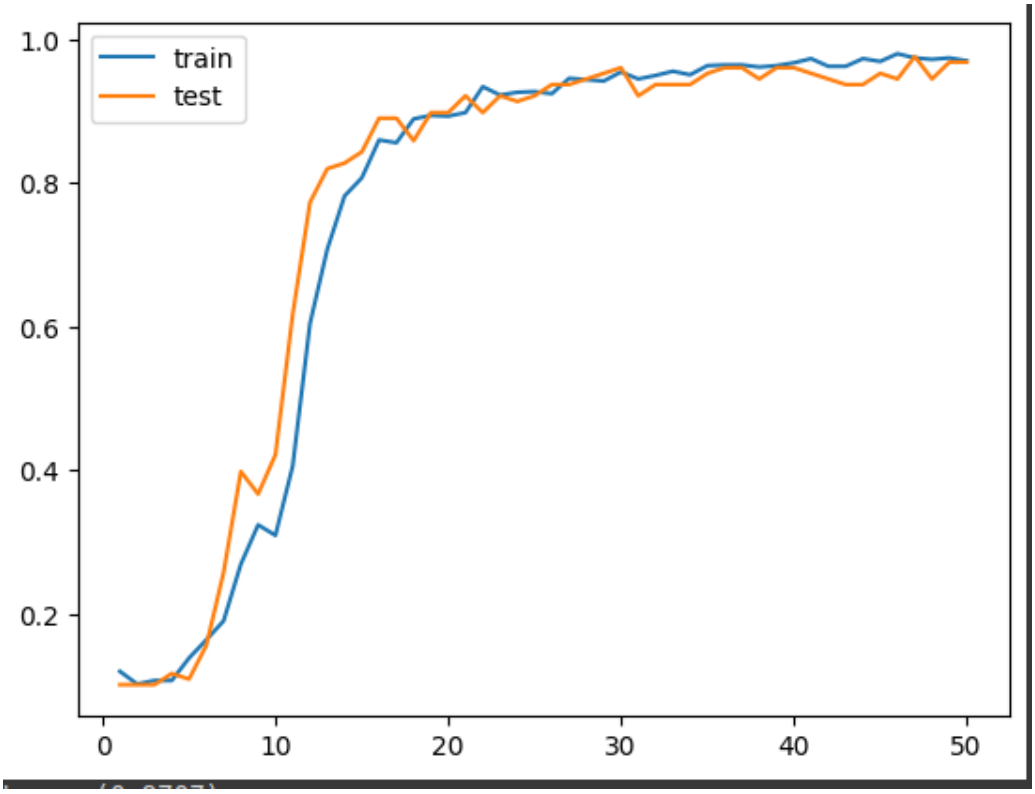

Рис. 3

На рисунке 3 приведен график точности модели LeNet с оптимизатором Adam и скоростью обучения 0.001.

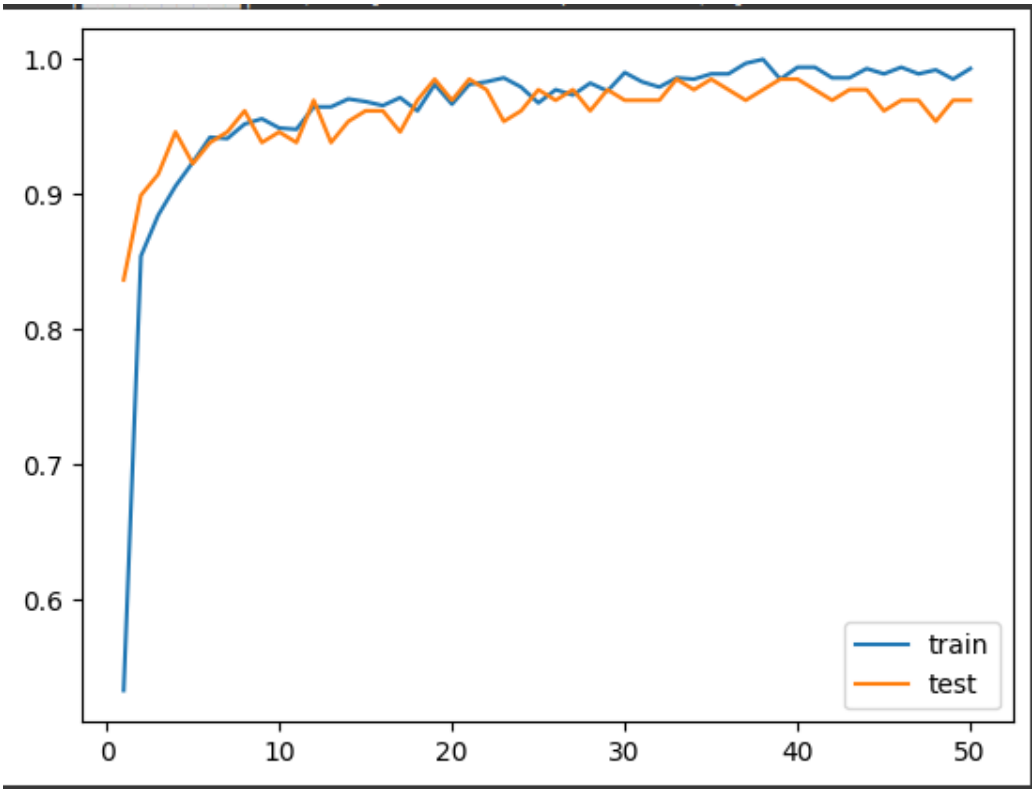

Рис. 4

На рисунке 5 приведен график точности модели MiniVGG с оптимизатором SGD и скоростью обучения 0.01.

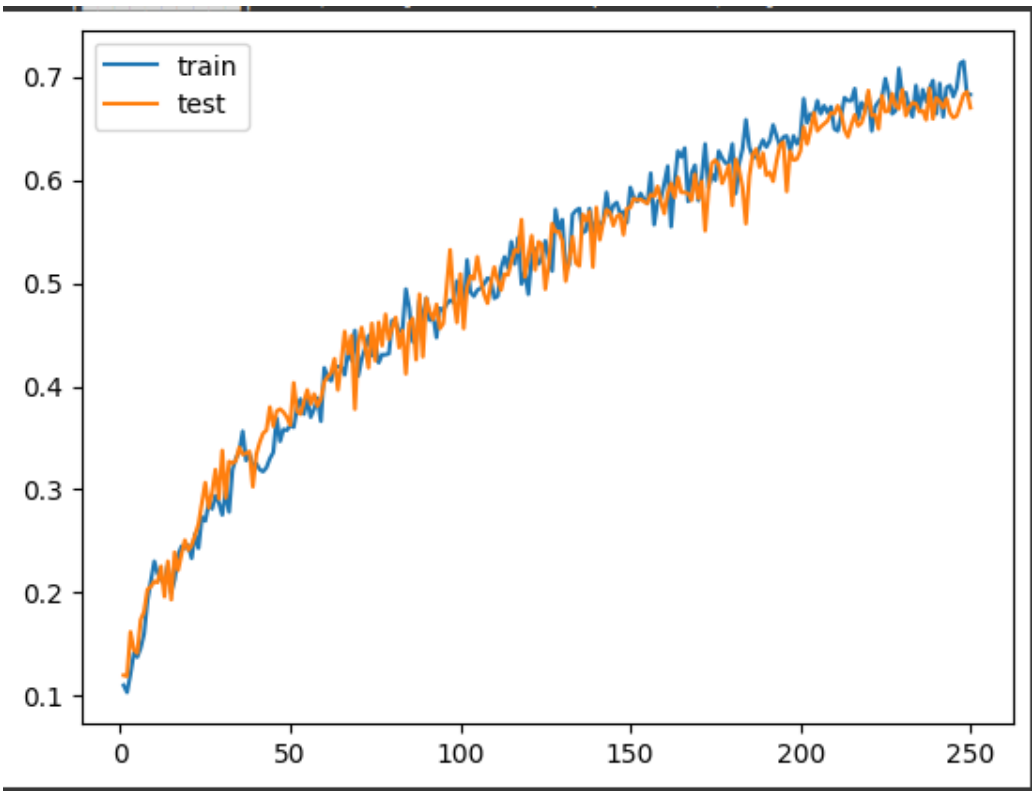

Рис. 5

На рисунке 6 приведен график точности модели MiniVGG с оптимизатором Adadelta и скоростью обучения 0.1.

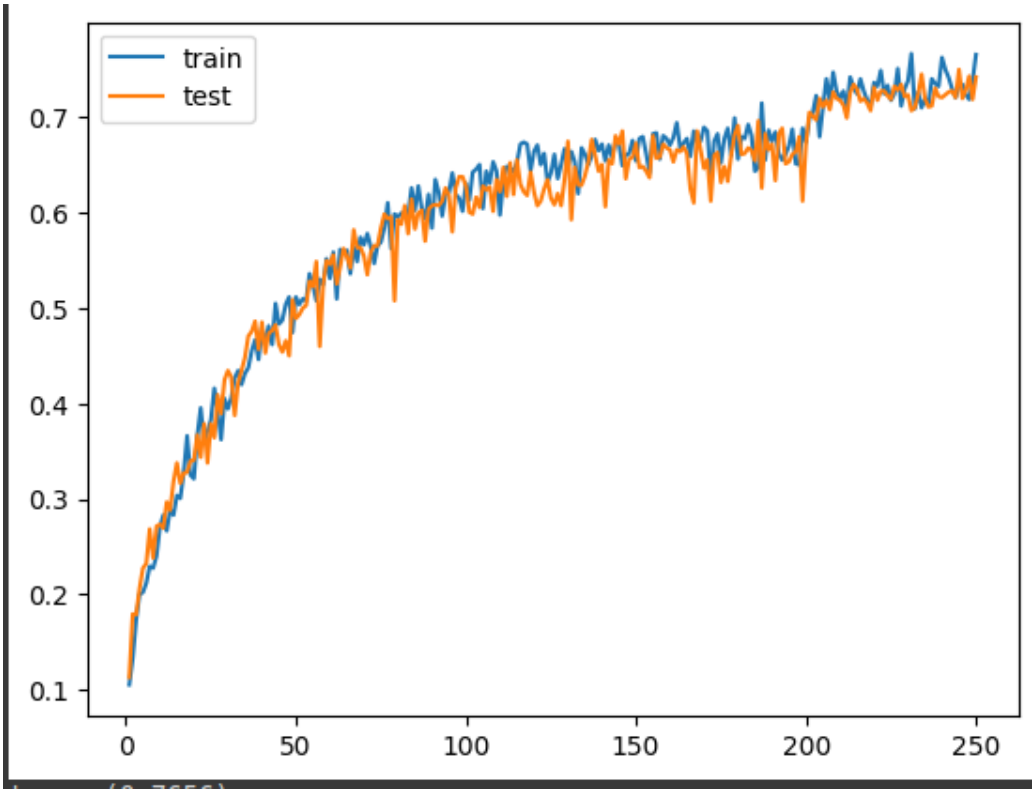

Рис. 6

На рисунке 7 приведен график точности модели MiniVGG с оптимизатором NAG и скоростью обучения 0.001.

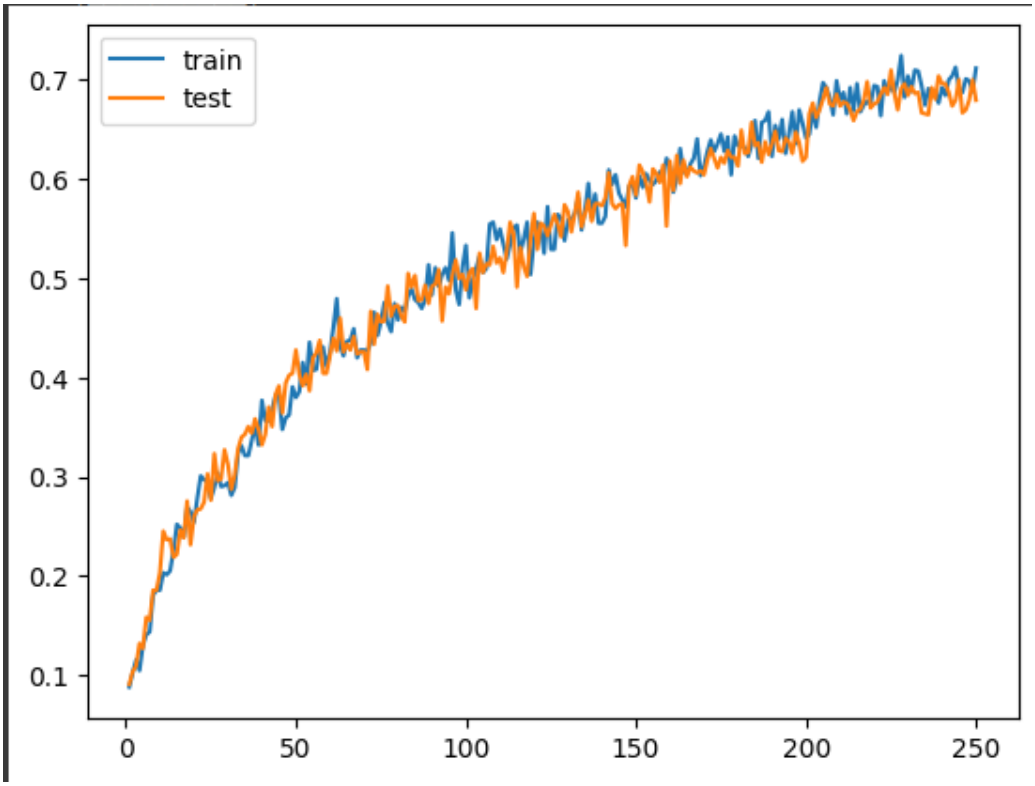

Рис. 7

На рисунке 8 приведен график точности модели MiniVGG с оптимизатором Adam и скоростью обучения 0.0001.

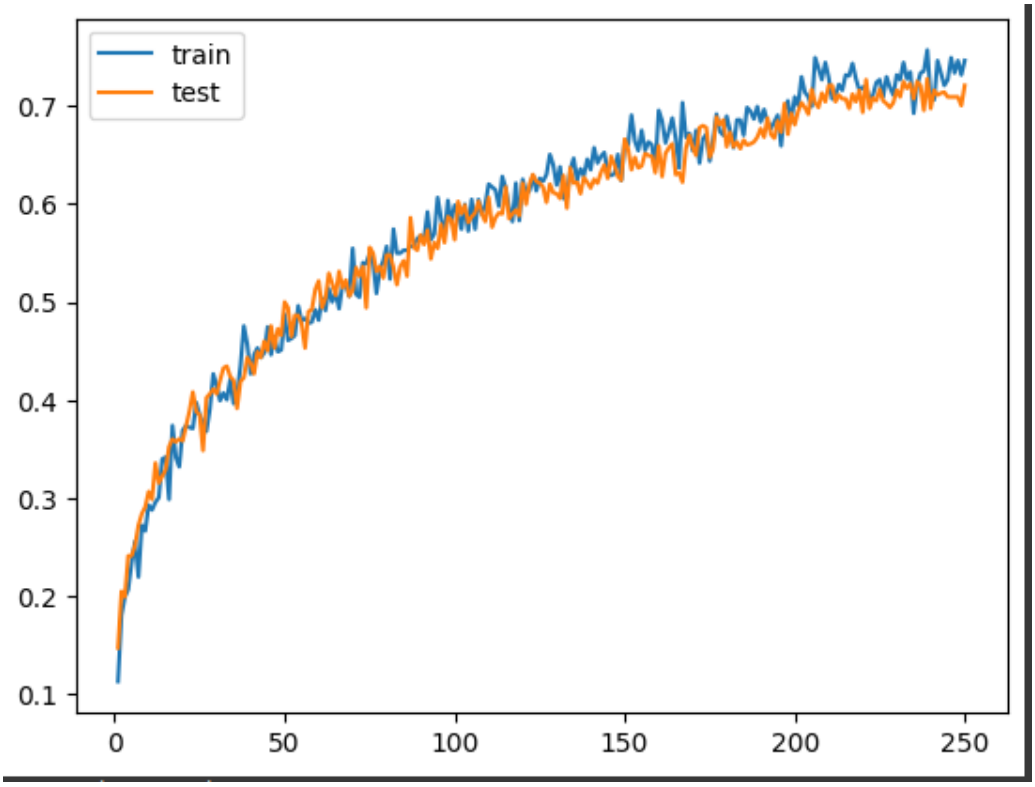

Рис. 8

На рисунке 9 приведен график точности модели ResNet с оптимизатором SGD и скоростью обучения 0.0015.

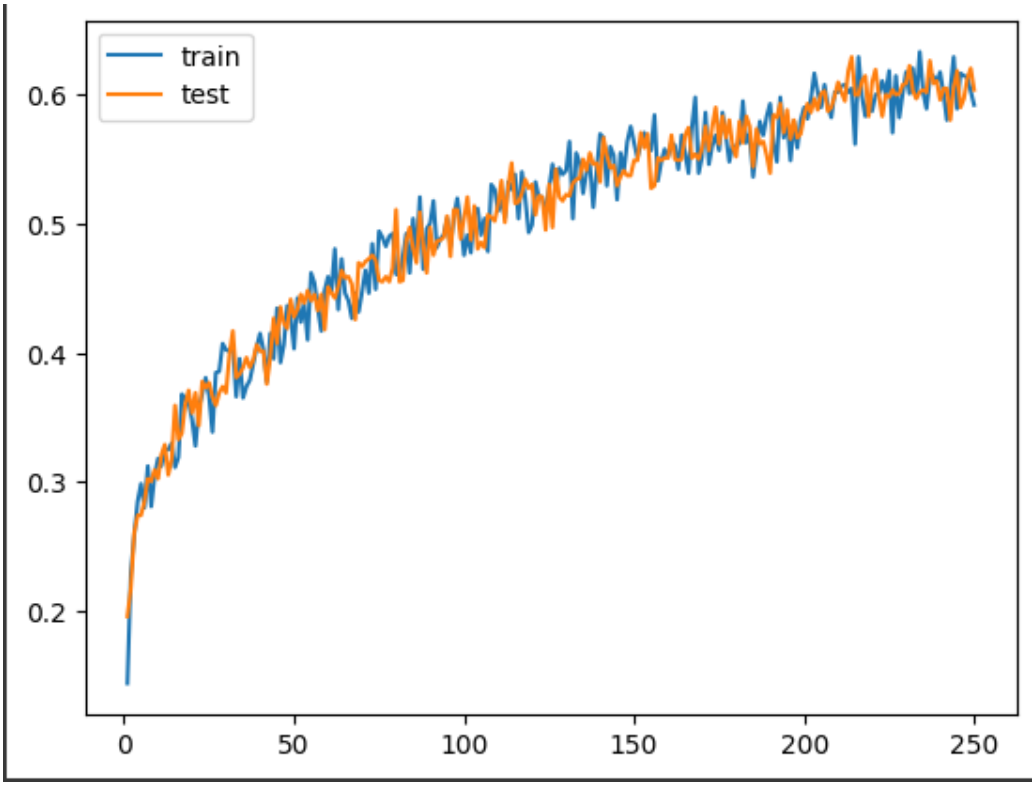

Рис. 9

На рисунке 10 приведен график точности модели ResNet с оптимизатором Adadelta и скоростью обучения 0.15.

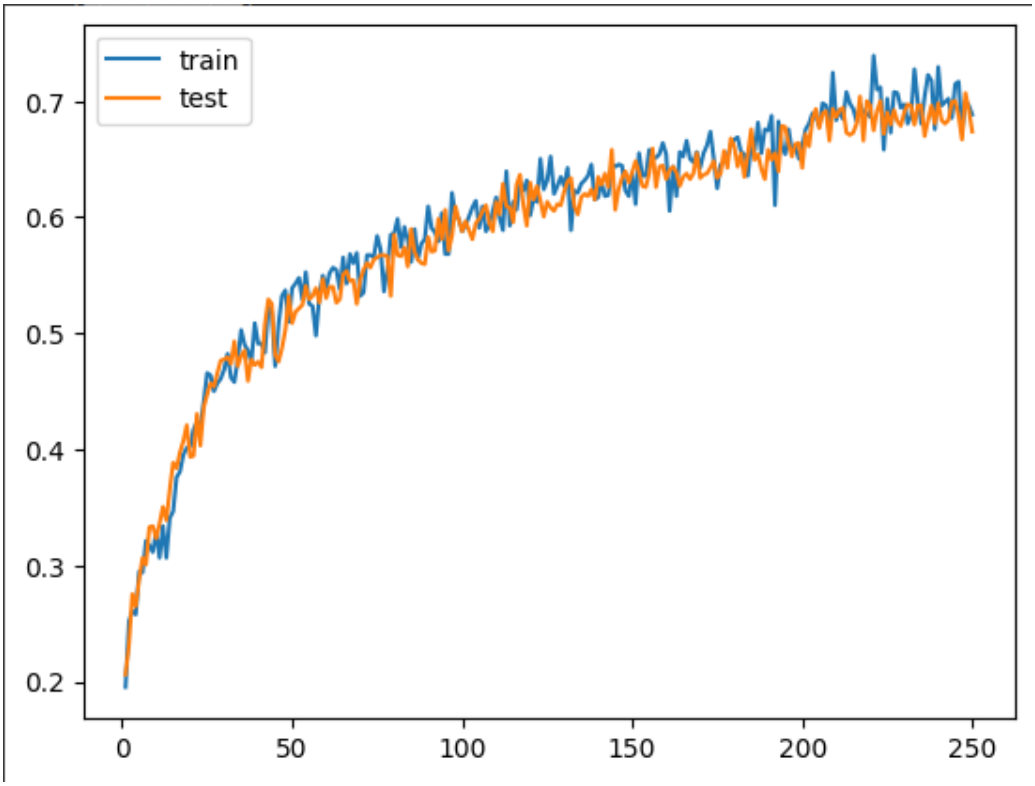

Рис. 10

На рисунке 11 приведен график точности модели ResNet с оптимизатором NAG и скоростью обучения 0.00015.

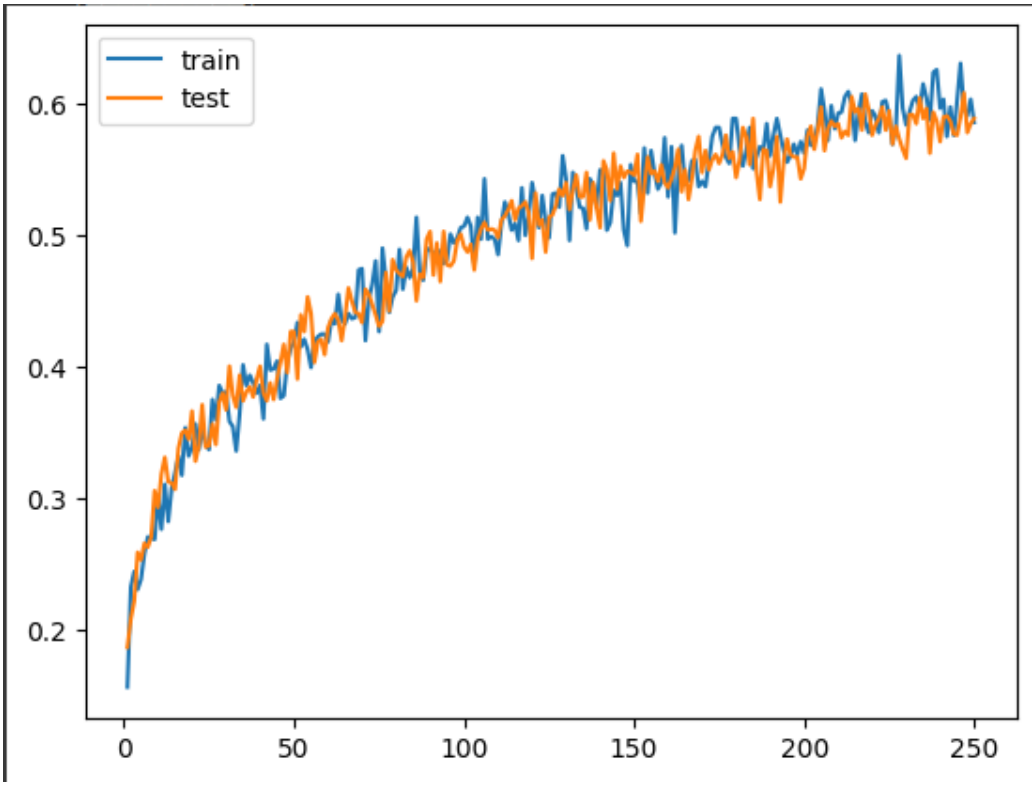

Рис. 11

На рисунке 12 приведен график точности модели ResNet с оптимизатором Adam и скоростью обучения 0.00015.

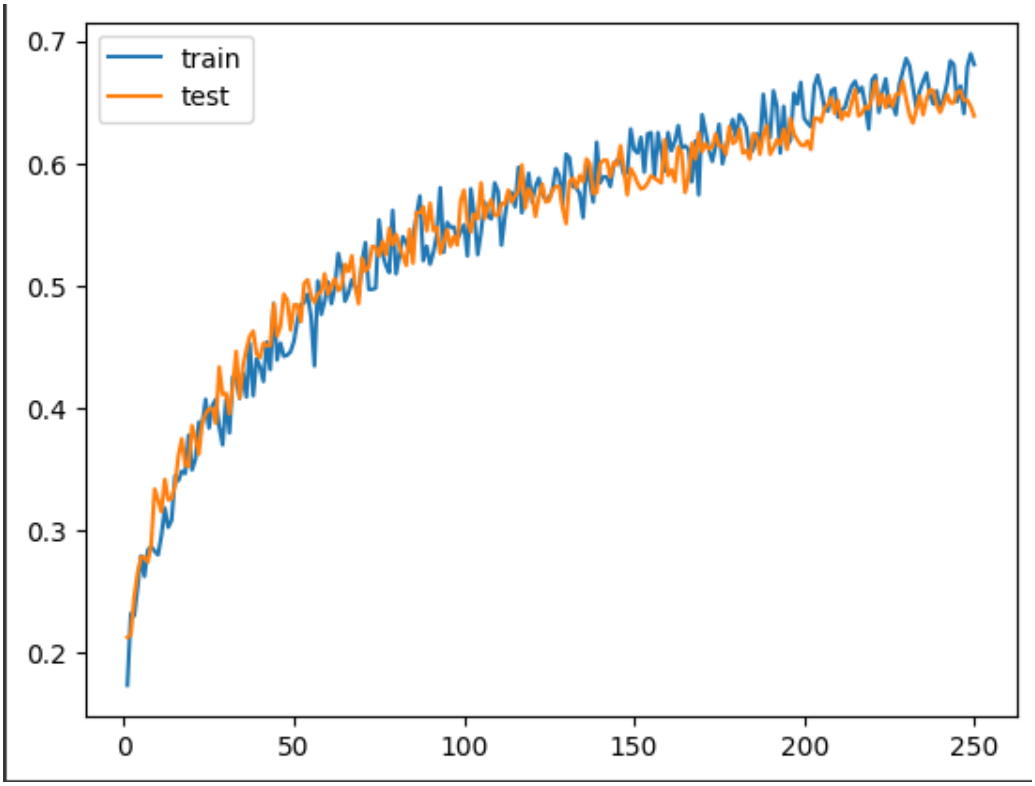

Рис. 12

В таблице 1 приведена вариация гиперпараметров для модели LeNet.

Таблица 1

Оптимизатор	Кол-во эпох	Скорость обучения	Точность
SGD	30	0.0055	0.9150
Adadelta	50	0.1	0.9668
NAG	50	0.001	0.9707
Adam	50	0.001	0.9922

В таблице 2 приведена вариация гиперпараметров для модели MiniVGG.

Таблица 2

Оптимизатор	Кол-во эпох	Скорость обучения	Dropout	Точность
SGD	250	0.01	0.5	0.6826
Adadelta	250	0.1	0.5	0.7656
NAG	250	0.001	0.5	0.7119
Adam	250	0.0001	0.5	0.7461

В таблице 3 приведена вариация гиперпараметров для модели ResNet.

Таблица 3

Оптимизатор	Кол-во эпох	Скорость обучения	Dropout	Точность
SGD	250	0.0015	0.5	0.5918
Adadelta	250	0.15	0.5	0.6807
NAG	250	0.00015	0.5	0.5859
Adam	250	0.00015	0.5	0.6885

4 Выводы

В рамках данной лабораторной работы с помощью популярного фреймворка PyTorch были реализованы следующие архитектуры сверточных нейронных сетей: LeNet, MiniVGG, ResNet. Среди экспериментов с LeNet лучшей оказалась модель с оптимизатором Adam и скоростью обучения 0.001. Среди экспериментов с MiniVGG лучшей оказалась модель с оптимизатором Adadelta и скоростью обучения 0.1. Среди экспериментов с ResNet лучшей оказалась модель с оптимизатором Adam и скоростью обучения 0.15.