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 (CNN)»

Студентка группы ИУ9-72Б Самохвалова П. С.

Преподаватель Каганов Ю. Т.

Москва 2023

1 Задание

1. Реализовать нейронную сеть LeNet. В качестве базы данных использовать MNIST. Применить оптимизаторы SGD, AdaDelta, NAG, Adam.

2. Реализовать нейронную сеть VGG16. В качестве базы данных использовать CIFAR-10. Применить оптимизаторы SGD, AdaDelta, NAG, Adam.

3. Реализовать нейронную сеть ResNet (34). В качестве базы данных использовать ImageNet. Применить оптимизаторы SGD, AdaDelta, NAG, Adam.

2 Практическая реализация

Исходный код программы представлен в листингах 1- 3.

Листинг 1: LeNet

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torch.nn.functional as F
import matplotlib.pyplot as plt


transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])


trainset = torchvision.datasets.MNIST(root='./data', train=True,
    download=True, transform=transform)
testset = torchvision.datasets.MNIST(root='./data', train=False,
    download=True, transform=transform)

trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
testloader = DataLoader(testset, batch_size=64, shuffle=False)


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

```
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16*4*4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        x = x.view(-1, 16*4*4)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


model = LeNet()

criterion = nn.CrossEntropyLoss()

optimizer_sgd = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
optimizer_adadelta = optim.Adadelta(model.parameters())
optimizer_nag = optim.SGD(model.parameters(), lr=0.01, momentum=0.9,
    nesterov=True)
optimizer_adam = optim.Adam(model.parameters())


def train_test_model(optimizer, name):
    model.train()
    losses = []
    for epoch in range(10):
        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            inputs, labels = data
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        epoch_loss = running_loss / len(trainloader)
        losses.append(epoch_loss)
        print(f"{name} - Epoch {epoch + 1} loss: {epoch_loss}")
```

```python

    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            images, labels = data
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    print(f"{name} - Accuracy: {accuracy}%")

    return losses


sgd_losses = train_test_model(optimizer_sgd, "SGD")
adadelta_losses = train_test_model(optimizer_adadelta, "AdaDelta")
nag_losses = train_test_model(optimizer_nag, "NAG")
adam_losses = train_test_model(optimizer_adam, "Adam")


epochs = range(1, 11)
plt.plot(epochs, sgd_losses, label='SGD')
plt.plot(epochs, adadelta_losses, label='AdaDelta')
plt.plot(epochs, nag_losses, label='NAG')
plt.plot(epochs, adam_losses, label='Adam')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss function dependence on the number of epochs for each optimizer')
plt.legend()
plt.show()
```

Листинг 2: VGG16

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt

model = torchvision.models.vgg16(pretrained=False)

num_classes = 10
```

```
model.classifier[6] = nn.Linear(4096, num_classes)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)

transform = transforms.Compose(
    [
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ]
)

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4, shuffle=False, num_workers=2)

optimizers = {
    "SGD": optim.SGD(model.parameters(), lr=0.001, momentum=0.9),
    "AdaDelta": optim.Adadelta(model.parameters(), lr=0.01),
    "NAG": optim.SGD(model.parameters(), lr=0.001, momentum=0.9, nesterov=True),
    "Adam": optim.Adam(model.parameters(), lr=0.00001)
}

for optimizer_name, optimizer in optimizers.items():
    criterion = nn.CrossEntropyLoss()
    epochs = 5

    optimizer_losses = [0] * epochs

    for epoch in range(epochs):
        running_loss = 0
        for i, data in enumerate(trainloader, 0):
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()

            outputs = model(inputs)
            loss = criterion(outputs, labels)
```

```
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            optimizer_losses[epoch] += loss.item()

            if i % 2000 == 1999:
                print(f"[{optimizer_name}, {epoch + 1}, {i + 1}] loss: {running_loss / 2000}")
                running_loss = 0.0

        optimizer_losses[epoch] /= len(trainloader)

    correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print(f"Accuracy of the network with {optimizer_name} optimizer: {100 * correct / total}%")

    plt.plot(range(1, epochs + 1), optimizer_losses, label=optimizer_name)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Loss function dependence on the number of epochs for optimizer ' + optimizer_name)
    plt.legend()
    plt.show()
```

Листинг 3: ResNet (34)

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
import matplotlib.pyplot as plt

```

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


transform = transforms.Compose([
     transforms.Resize(256),
     transforms.CenterCrop(224),
     transforms.ToTensor(),
     transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])


train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)


train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=64, shuffle=False)


model = models.resnet34(pretrained=False)
model.to(device)


criterion = nn.CrossEntropyLoss()


accuracies = {}


optimizers = ['SGD', 'Adadelta', 'NAG', 'Adam']
losses = {optimizer_name: [] for optimizer_name in optimizers}

for optimizer_name in optimizers:
    if optimizer_name == 'SGD':
        optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
    elif optimizer_name == 'Adadelta':
        optimizer = optim.Adadelta(model.parameters())
    elif optimizer_name == 'NAG':
        optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9, nesterov=True)
```

```
    elif optimizer_name == 'Adam':
        optimizer = optim.Adam(model.parameters(), lr=0.001)

    for epoch in range(5):
        model.train()
        running_loss = 0.0
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        print(f"Epoch {epoch+1}, Optimizer: {optimizer_name}, Loss: {running_loss / len(train_loader)}")
        losses[optimizer_name].append(running_loss / len(train_loader))

    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    accuracies[optimizer_name] = accuracy
    print(f'Accuracy of the network on the test images with {optimizer_name} optimizer: {accuracy:.2f}%')

    plt.plot(range(1, 6), losses[optimizer_name], label=optimizer_name)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Loss function dependence on the number of epochs for optimizer ' + optimizer_name)
    plt.legend()
    plt.show()


print("Accuracies for different optimizers:")
for optimizer_name, accuracy in accuracies.items():
    print(f"{optimizer_name}: {accuracy:.2f}%")
```

3 Результаты

Результаты работы программы представлены на рисунках 1- 19.

```
SGD - Epoch 1 loss: 0.04797668762797396
SGD - Epoch 2 loss: 0.036027234870811865
SGD - Epoch 3 loss: 0.028856778872211893
SGD - Epoch 4 loss: 0.024484700602793 9
SGD - Epoch 5 loss: 0.021115803868930996
SGD - Epoch 6 loss: 0.01876699934248874
SGD - Epoch 7 loss: 0.014785314569834805
SGD - Epoch 8 loss: 0.014005225215688177
SGD - Epoch 9 loss: 0.010436913913792856
SGD - Epoch 10 loss: 0.010134605349389519
SGD - Accuracy: 98.86%
AdaDelta - Epoch 1 loss: 0.022558863759092957
AdaDelta - Epoch 2 loss: 0.019058083317939987
AdaDelta - Epoch 3 loss: 0.016210800011888387
AdaDelta - Epoch 4 loss: 0.013096760662922034
AdaDelta - Epoch 5 loss: 0.012577806868266005
AdaDelta - Epoch 6 loss: 0.010397543301558959
AdaDelta - Epoch 7 loss: 0.009481603296815095
AdaDelta - Epoch 8 loss: 0.00808082507300579
AdaDelta - Epoch 9 loss: 0.006468135072998011
AdaDelta - Epoch 10 loss: 0.005467295557532801
AdaDelta - Accuracy: 98.93%
```

Рис. 1 — Результат работы нейронной сети LeNet с оптимизаторами SGD и AdaDelta

```
NAG - Epoch 1 loss: 0.007875829274970 22
NAG - Epoch 2 loss: 0.004608031559518695
NAG - Epoch 3 loss: 0.0022396754083366184
NAG - Epoch 4 loss: 0.003136958142601099
NAG - Epoch 5 loss: 0.003943243743814708
NAG - Epoch 6 loss: 0.0016330818259314116
NAG - Epoch 7 loss: 0.00037742435931199456
NAG - Epoch 8 loss: 0.0006183253060741904
NAG - Epoch 9 loss: 0.00020005558085133064
NAG - Epoch 10 loss: 4.119652412146766e-05
NAG - Accuracy: 99.16%
Adam - Epoch 1 loss: 0.013740929395652 12
Adam - Epoch 2 loss: 0.011992538895982336
Adam - Epoch 3 loss: 0.009622250434174022
Adam - Epoch 4 loss: 0.008653295581118986
Adam - Epoch 5 loss: 0.009547748116000702
Adam - Epoch 6 loss: 0.0103140324600208
Adam - Epoch 7 loss: 0.007459128912773552
Adam - Epoch 8 loss: 0.008228338507701181
Adam - Epoch 9 loss: 0.006582019857332815
Adam - Epoch 10 loss: 0.006534842348275204
Adam - Accuracy: 98.93%
```

Рис. 2 — Результат работы нейронной сети LeNet с оптимизаторами NAG и Adam

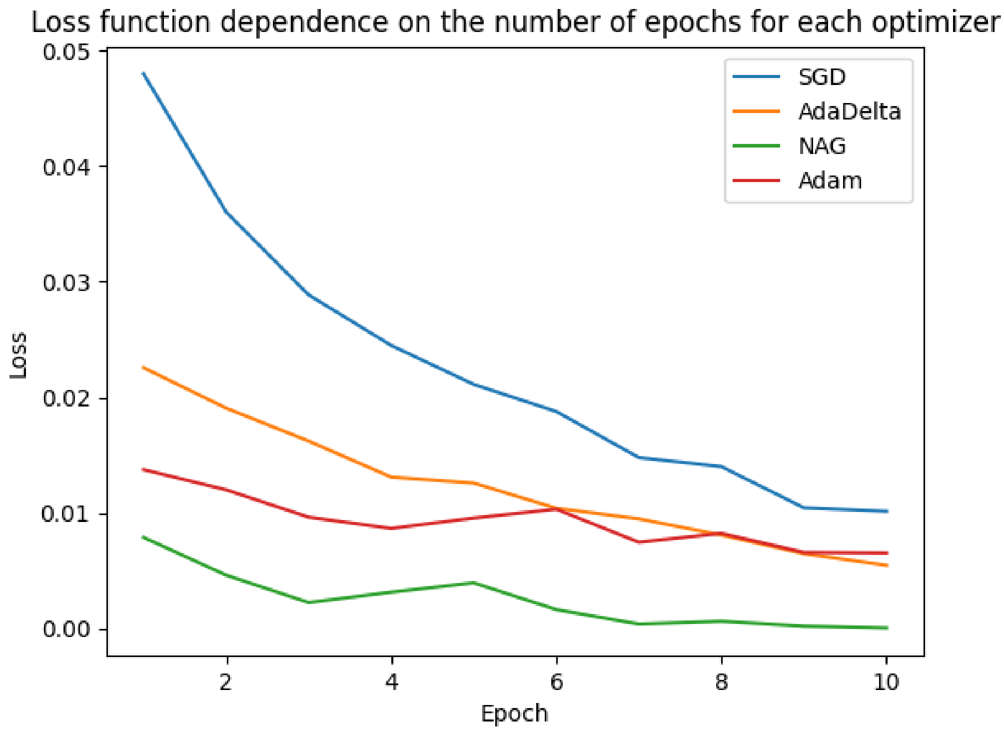


Рис. 3 — Результат работы нейронной сети LeNet с оптимизаторами SGD, AdaDelta, NAG и Adam

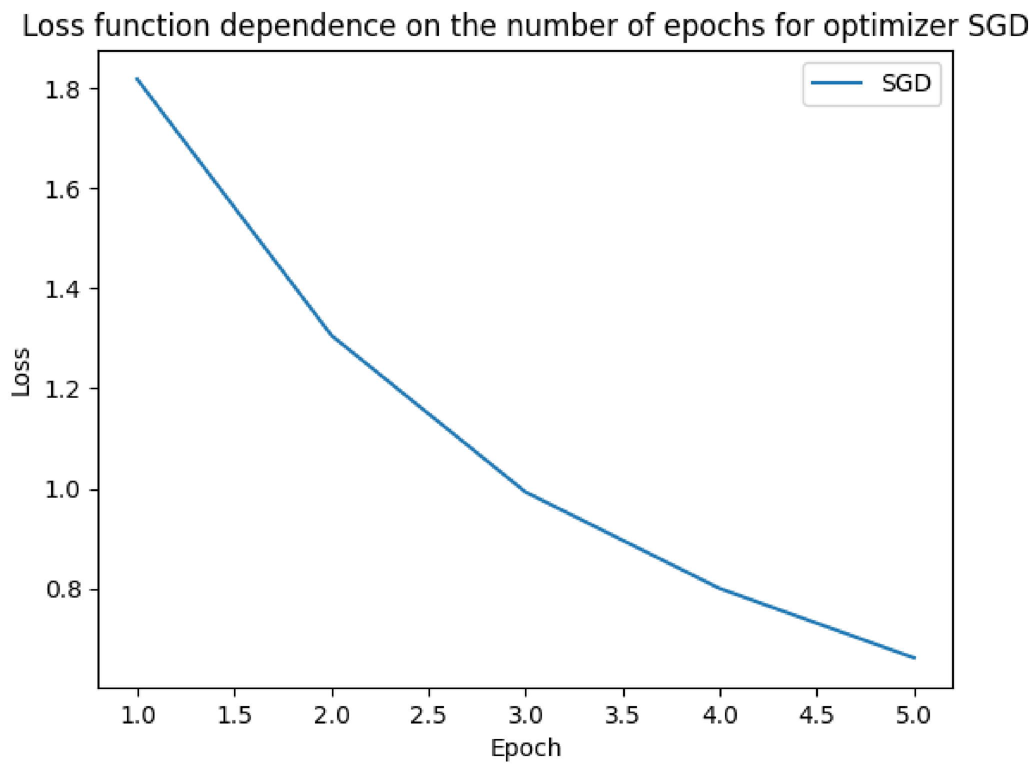
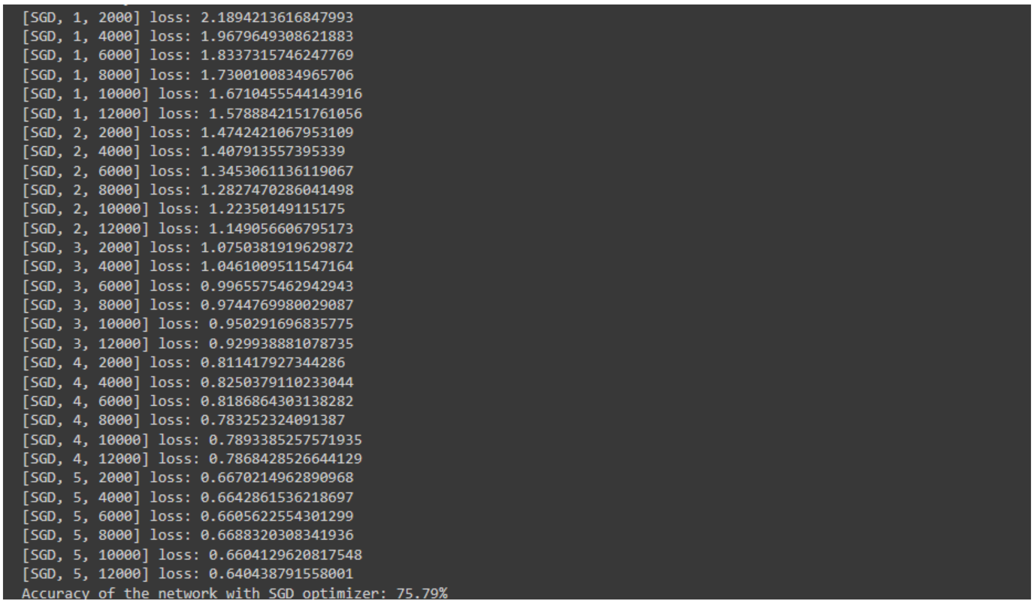


Рис. 4 — Результат работы нейронной сети VGG16 с оптимизатором SGD



```
[SGD, 1, 2000] loss: 2.1894213616847993
[SGD, 1, 4000] loss: 1.9679649308621883
[SGD, 1, 6000] loss: 1.8337315746247769
[SGD, 1, 8000] loss: 1.7300100834965706
[SGD, 1, 10000] loss: 1.6710455544143916
[SGD, 1, 12000] loss: 1.5788842151761056
[SGD, 2, 2000] loss: 1.4742421067953109
[SGD, 2, 4000] loss: 1.407913557395339
[SGD, 2, 6000] loss: 1.3453061136119067
[SGD, 2, 8000] loss: 1.2827470286041498
[SGD, 2, 10000] loss: 1.22350149115175
[SGD, 2, 12000] loss: 1.149056606795173
[SGD, 3, 2000] loss: 1.0750381919629872
[SGD, 3, 4000] loss: 1.0461009511547164
[SGD, 3, 6000] loss: 0.9965575462942943
[SGD, 3, 8000] loss: 0.9744769980029087
[SGD, 3, 10000] loss: 0.950291696835775
[SGD, 3, 12000] loss: 0.929938881078735
[SGD, 4, 2000] loss: 0.811417927344286
[SGD, 4, 4000] loss: 0.8250379110233044
[SGD, 4, 6000] loss: 0.8186864303138282
[SGD, 4, 8000] loss: 0.783252324091387
[SGD, 4, 10000] loss: 0.7893385257571935
[SGD, 4, 12000] loss: 0.7868428526644129
[SGD, 5, 2000] loss: 0.6670214962890968
[SGD, 5, 4000] loss: 0.6642861536218697
[SGD, 5, 6000] loss: 0.6605622554301299
[SGD, 5, 8000] loss: 0.6688320308341936
[SGD, 5, 10000] loss: 0.6604129620817548
[SGD, 5, 12000] loss: 0.640438791558001
Accuracy of the network with SGD optimizer: 75.79%
```

Рис. 5 — Результат работы нейронной сети VGG16 с оптимизатором SGD

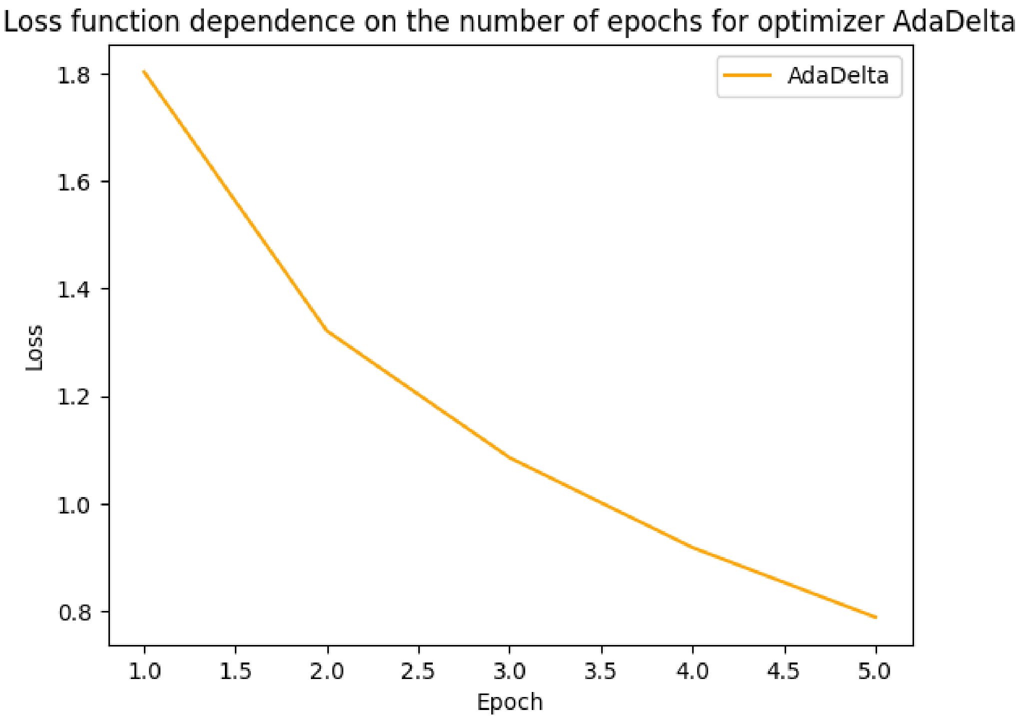


Рис. 6 — Результат работы нейронной сети VGG16 с оптимизатором AdaDelta

4 Выводы

В результате выполнения лабораторной работы на PyTorch были реализованы нейронные сети LeNet, VGG16, ResNet (34) с оптимизаторами SGD, AdaDelta, NAG, Adam.

```
[AdaDelta, 1, 2000] loss: 2.2499634036421776
[AdaDelta, 1, 4000] loss: 1.9987514218091964
[AdaDelta, 1, 6000] loss: 1.7931090692132712
[AdaDelta, 1, 8000] loss: 1.6870725436359644
[AdaDelta, 1, 10000] loss: 1.6093845569044352
[AdaDelta, 1, 12000] loss: 1.5544235248938203
[AdaDelta, 2, 2000] loss: 1.4484482859969139
[AdaDelta, 2, 4000] loss: 1.3755523818135262
[AdaDelta, 2, 6000] loss: 1.3382227805368603
[AdaDelta, 2, 8000] loss: 1.2961661068052053
[AdaDelta, 2, 10000] loss: 1.2599063619673252
[AdaDelta, 2, 12000] loss: 1.2402095287442207
[AdaDelta, 3, 2000] loss: 1.149511059306562
[AdaDelta, 3, 4000] loss: 1.0996184348287061
[AdaDelta, 3, 6000] loss: 1.1147168178567664
[AdaDelta, 3, 8000] loss: 1.0707872729040682
[AdaDelta, 3, 10000] loss: 1.0386965093459002
[AdaDelta, 3, 12000] loss: 1.059362464456819
[AdaDelta, 4, 2000] loss: 0.9536775740922895
[AdaDelta, 4, 4000] loss: 0.9114514478055061
[AdaDelta, 4, 6000] loss: 0.9339479059856385
[AdaDelta, 4, 8000] loss: 0.9156570988511666
[AdaDelta, 4, 10000] loss: 0.8876173911169172
[AdaDelta, 4, 12000] loss: 0.9052881841888738
[AdaDelta, 5, 2000] loss: 0.7754616069906916
[AdaDelta, 5, 4000] loss: 0.7705171618932727
[AdaDelta, 5, 6000] loss: 0.7868890565221663
[AdaDelta, 5, 8000] loss: 0.7946296058123989
[AdaDelta, 5, 10000] loss: 0.803992246422873
[AdaDelta, 5, 12000] loss: 0.794486996242762
Accuracy of the network with AdaDelta optimizer: 70.08%
```

Рис. 7 — Результат работы нейронной сети VGG16 с оптимизатором AdaDelta

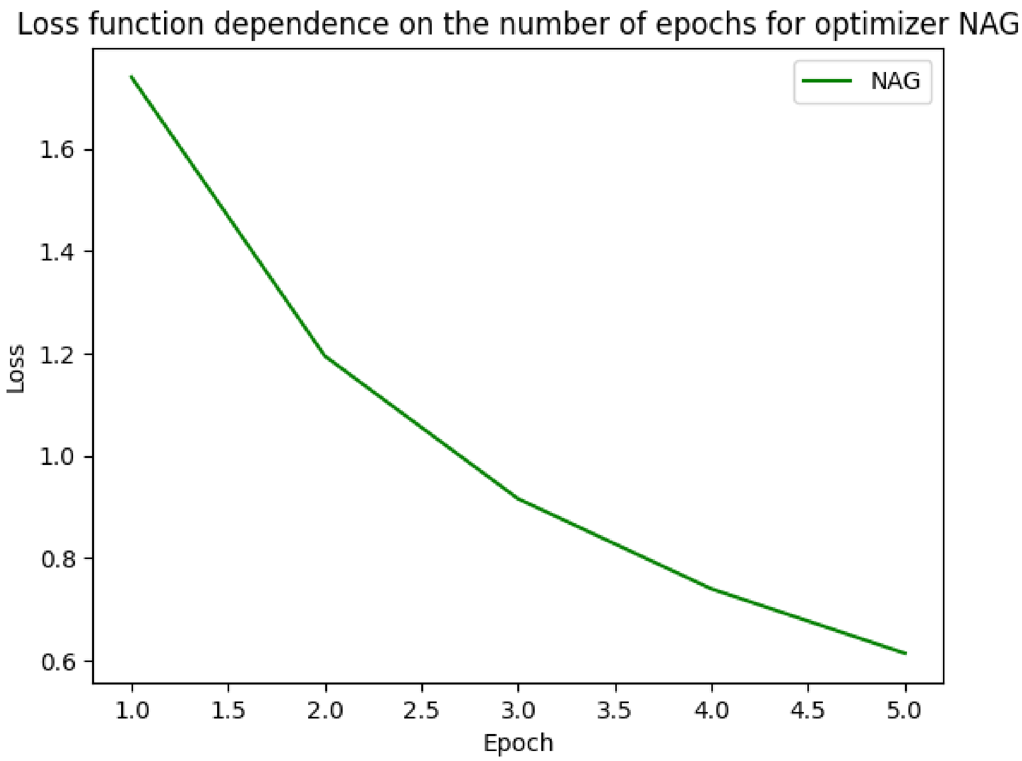
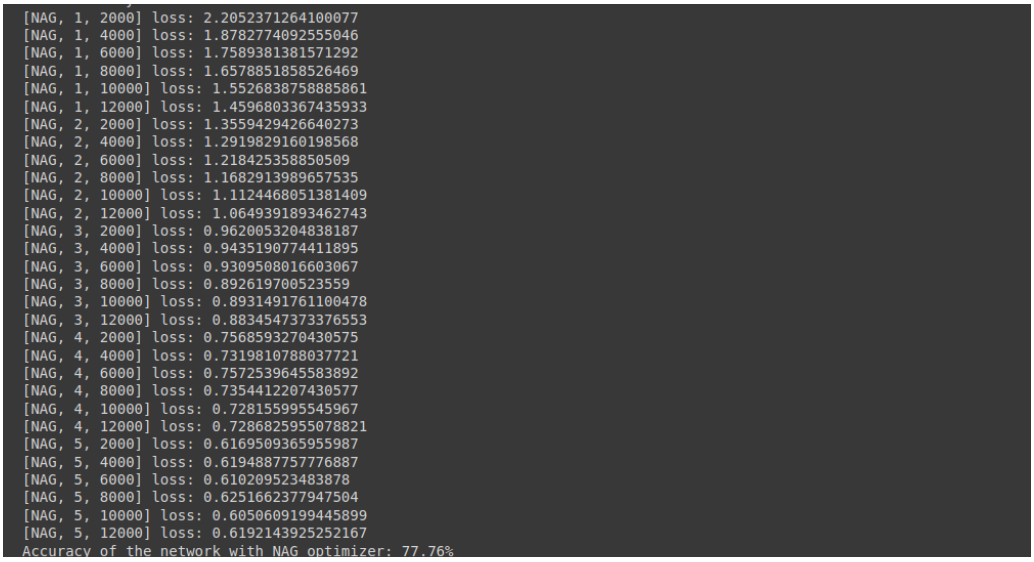


Рис. 8 — Результат работы нейронной сети VGG16 с оптимизатором NAG



```
[NAG, 1, 2000] loss: 2.2052371264100077
[NAG, 1, 4000] loss: 1.8782774092555046
[NAG, 1, 6000] loss: 1.7589381381571292
[NAG, 1, 8000] loss: 1.6578851858526469
[NAG, 1, 10000] loss: 1.5526838758885861
[NAG, 1, 12000] loss: 1.4596803367435933
[NAG, 2, 2000] loss: 1.3559429426640273
[NAG, 2, 4000] loss: 1.2919829160198568
[NAG, 2, 6000] loss: 1.218425358850509
[NAG, 2, 8000] loss: 1.1682913989657535
[NAG, 2, 10000] loss: 1.1124468051381409
[NAG, 2, 12000] loss: 1.0649391893462743
[NAG, 3, 2000] loss: 0.9620053204838187
[NAG, 3, 4000] loss: 0.9435190774411895
[NAG, 3, 6000] loss: 0.9309508016603067
[NAG, 3, 8000] loss: 0.892619700523559
[NAG, 3, 10000] loss: 0.8931491761100478
[NAG, 3, 12000] loss: 0.8834547373376553
[NAG, 4, 2000] loss: 0.7568593270430575
[NAG, 4, 4000] loss: 0.7319810788037721
[NAG, 4, 6000] loss: 0.7572539645583892
[NAG, 4, 8000] loss: 0.7354412207430577
[NAG, 4, 10000] loss: 0.728155995545967
[NAG, 4, 12000] loss: 0.7286825955078821
[NAG, 5, 2000] loss: 0.6169509365955987
[NAG, 5, 4000] loss: 0.6194887757776887
[NAG, 5, 6000] loss: 0.610209523483878
[NAG, 5, 8000] loss: 0.6251662377947504
[NAG, 5, 10000] loss: 0.6050609199445899
[NAG, 5, 12000] loss: 0.6192143925252167
Accuracy of the network with NAG optimizer: 77.76%
```

Рис. 9 — Результат работы нейронной сети VGG16 с оптимизатором NAG

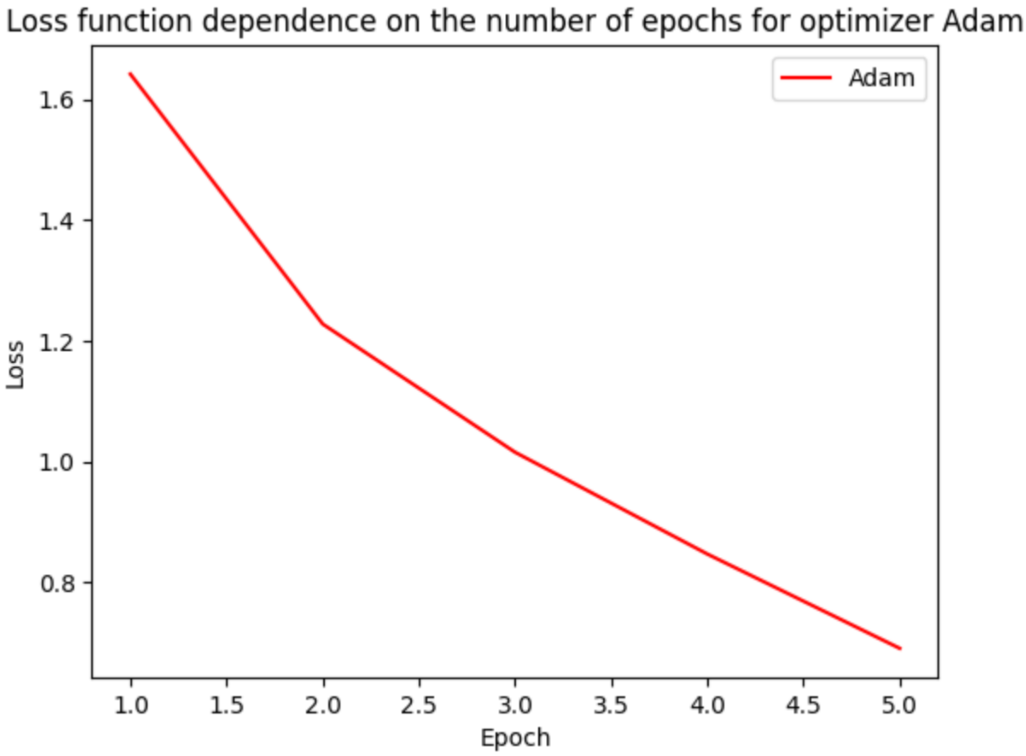


Рис. 10 — Результат работы нейронной сети VGG16 с оптимизатором Adam

```
[Adam, 1, 2000] loss: 2.062741495132446
[Adam, 1, 4000] loss: 1.7933170551359654
[Adam, 1, 6000] loss: 1.6396787562668325
[Adam, 1, 8000] loss: 1.5327414567321538
[Adam, 1, 10000] loss: 1.4783946781754493
[Adam, 1, 12000] loss: 1.4157332792580128
[Adam, 2, 2000] loss: 1.3190140459835529
[Adam, 2, 4000] loss: 1.275151315279305
[Adam, 2, 6000] loss: 1.2704246242195367
[Adam, 2, 8000] loss: 1.2003670376949012
[Adam, 2, 10000] loss: 1.1730571854375302
[Adam, 2, 12000] loss: 1.1478230611942708
[Adam, 3, 2000] loss: 1.0443953196927904
[Adam, 3, 4000] loss: 1.0508131722975522
[Adam, 3, 6000] loss: 1.0266857065763324
[Adam, 3, 8000] loss: 1.002074948183261
[Adam, 3, 10000] loss: 0.9888059260174632
[Adam, 3, 12000] loss: 0.9903911812938749
[Adam, 4, 2000] loss: 0.8600541572719812
[Adam, 4, 4000] loss: 0.8675965261529199
[Adam, 4, 6000] loss: 0.8381941680572927
[Adam, 4, 8000] loss: 0.8556119765313343
[Adam, 4, 10000] loss: 0.8459248103424907
[Adam, 4, 12000] loss: 0.8205485811084509
[Adam, 5, 2000] loss: 0.6895021487821359
[Adam, 5, 4000] loss: 0.6896921383730369
[Adam, 5, 6000] loss: 0.6925000499776798
[Adam, 5, 8000] loss: 0.6893276025557424
[Adam, 5, 10000] loss: 0.6943560376940295
[Adam, 5, 12000] loss: 0.6944746703179553
Accuracy of the network with Adam optimizer: 67.48%
```

Рис. 11 — Результат работы нейронной сети VGG16 с оптимизатором Adam

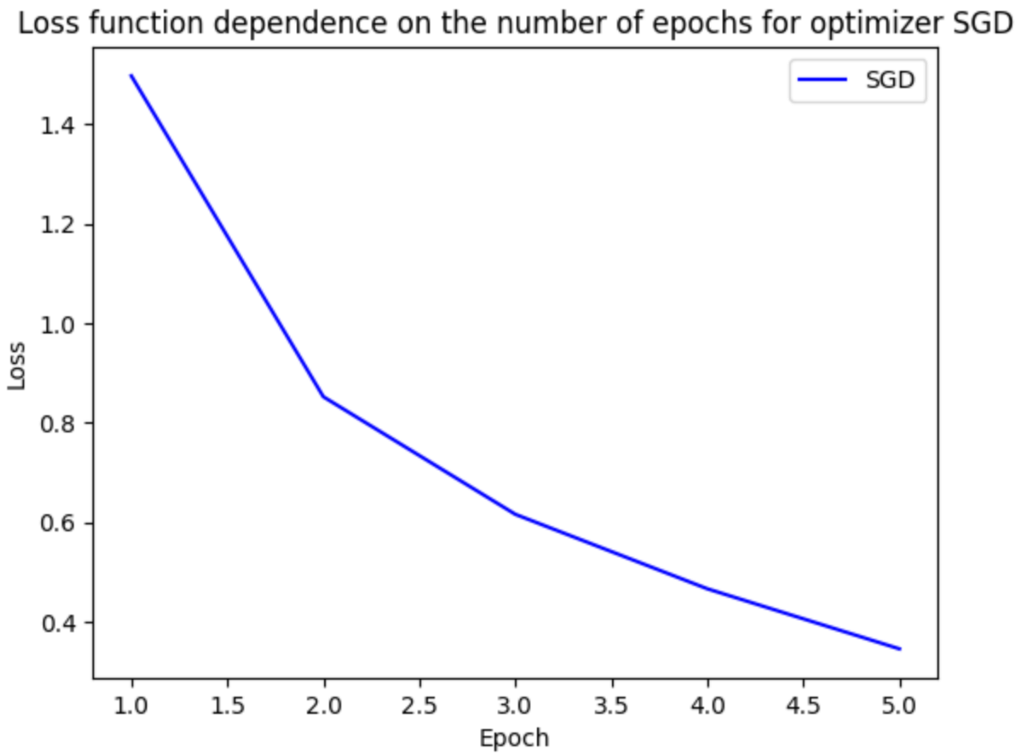


Рис. 12 — Результат работы нейронной сети ResNet (34) с оптимизатором SGD

```
Epoch 1, Optimizer: SGD, Loss: 1.4968267785923561
Epoch 2, Optimizer: SGD, Loss: 0.8519377305227167
Epoch 3, Optimizer: SGD, Loss: 0.6156465031606767
Epoch 4, Optimizer: SGD, Loss: 0.4660088890958625
Epoch 5, Optimizer: SGD, Loss: 0.3453148864114376
Accuracy of the network on the test images with SGD optimizer: 74.23%
```

Рис. 13 — Результат работы нейронной сети ResNet (34) с оптимизатором SGD

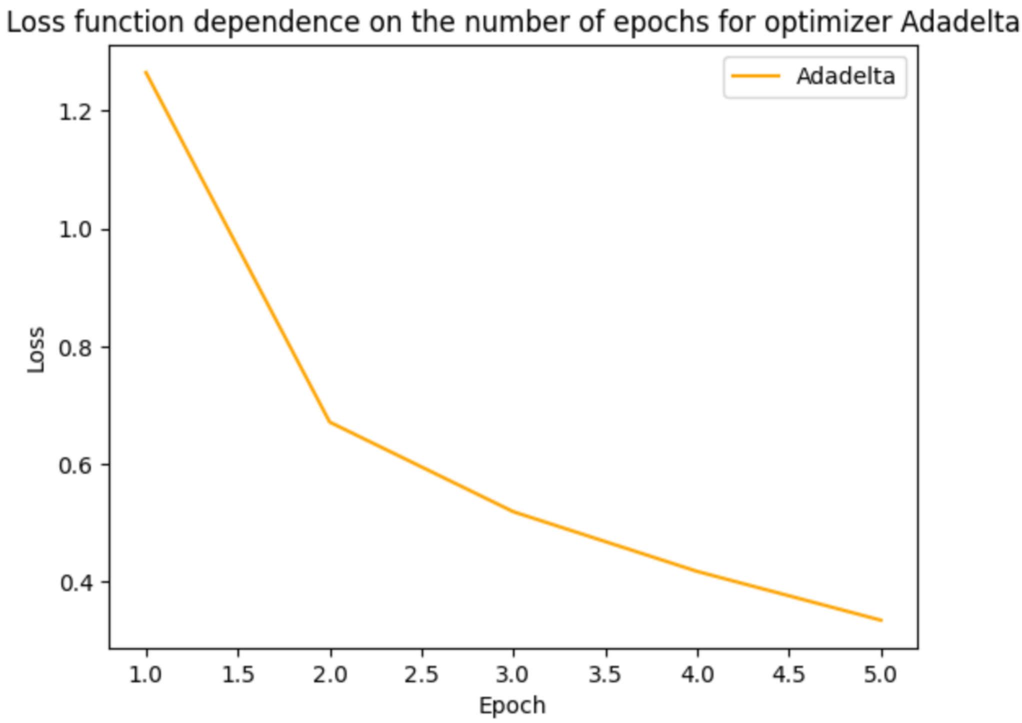
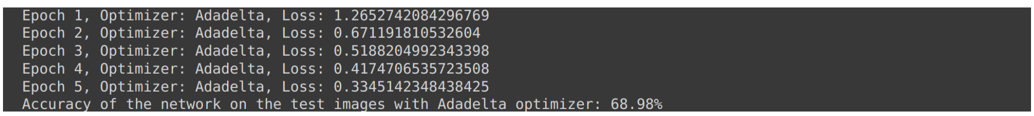


Рис. 14 — Результат работы нейронной сети ResNet (34) с оптимизатором AdaDelta



```
Epoch 1, Optimizer: Adadelta, Loss: 1.2652742084296769
Epoch 2, Optimizer: Adadelta, Loss: 0.671191810532604
Epoch 3, Optimizer: Adadelta, Loss: 0.5188204992343398
Epoch 4, Optimizer: Adadelta, Loss: 0.4174706535723508
Epoch 5, Optimizer: Adadelta, Loss: 0.3345142348438425
Accuracy of the network on the test images with Adadelta optimizer: 68.98%
```

Рис. 15 — Результат работы нейронной сети ResNet (34) с оптимизатором AdaDelta

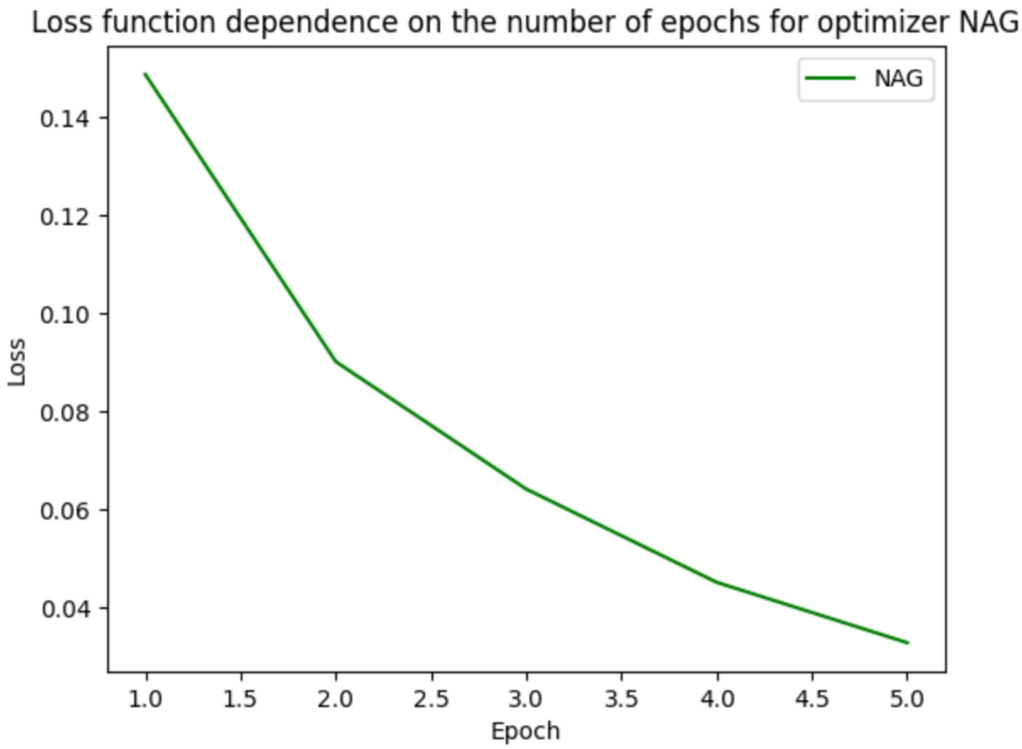
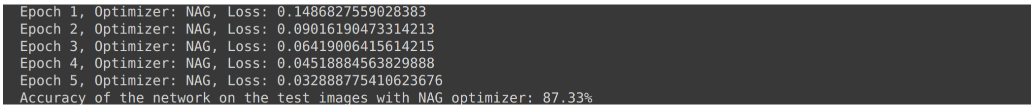


Рис. 16 — Результат работы нейронной сети ResNet (34) с оптимизатором NAG



```
Epoch 1, Optimizer: NAG, Loss: 0.1486827559028383
Epoch 2, Optimizer: NAG, Loss: 0.09016190473314213
Epoch 3, Optimizer: NAG, Loss: 0.06419006415614215
Epoch 4, Optimizer: NAG, Loss: 0.04518884563829888
Epoch 5, Optimizer: NAG, Loss: 0.032888775410623676
Accuracy of the network on the test images with NAG optimizer: 87.33%
```

Рис. 17 — Результат работы нейронной сети ResNet (34) с оптимизатором NAG

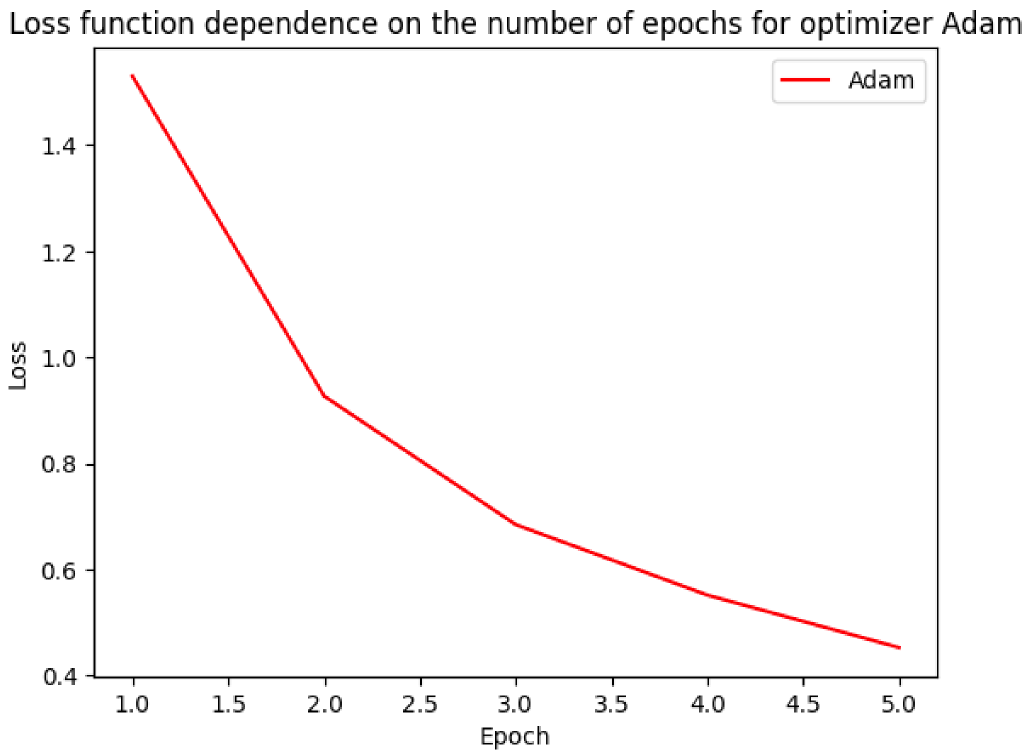


Рис. 18 — Результат работы нейронной сети ResNet (34) с оптимизатором Adam

```
Epoch 1, Optimizer: Adam, Loss: 1.529735531495965
Epoch 2, Optimizer: Adam, Loss: 0.926960490434371
Epoch 3, Optimizer: Adam, Loss: 0.6846035719298951
Epoch 4, Optimizer: Adam, Loss: 0.5517108656103958
Epoch 5, Optimizer: Adam, Loss: 0.45280647239721644
Accuracy of the network on the test images with Adam optimizer: 80.00%
```

Рис. 19 — Результат работы нейронной сети ResNet (34) с оптимизатором Adam