|  | **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 Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**Домашнее задание № 5**

**Сверточные нейронные сети (CNN)**

**ПО КУРСУ:**

***«Теория искусственных нейронных сетей»***

Студент *Караник А. А.*

Преподаватель *Каганов Ю. Т.*

*Москва, 2024 г.*

**ОГЛАВЛЕНИЕ**

[Цель работы 3](#_Toc184733432)

[Постановка задачи 3](#_Toc184733433)

[Практическая реализация 3](#_Toc184733434)

[Результаты 6](#_Toc184733435)

[Вывод 11](#_Toc184733436)

# Цель работы

1. Изучение сверточных нейронных сетей.
2. Программная реализация архитектур сверточных нейронных сетей.
3. Обучение нейронных сетей на распознавание изображений.

# Постановка задачи

1. Реализовать три модели CNN: LeNet, VGG16, ResNet (34).
2. Провести сравнительный анализ методов оптимизации (SGD, AdaDelta, NAG, Adam) для каждой модели.
3. Выполнить поиск оптимальных гиперпараметров для каждого метода оптимизации.

# Практическая реализация

Исходный текст программы на языке программирования Python:

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 fctl

import matplotlib.pyplot as plt

import torchvision.datasets as datasets

import torchvision.models as models

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 = fctl.relu(self.conv1(x))

        x = fctl.max\_pool2d(x, 2)

        x = fctl.relu(self.conv2(x))

        x = fctl.max\_pool2d(x, 2)

        x = x.view(-1, 16\*4\*4)

        x = fctl.relu(self.fc1(x))

        x = fctl.relu(self.fc2(x))

        x = self.fc3(x)

        return x

model = LeNet()

criterion = nn.CrossEntropyLoss()

optimizer\_sgd = optim.SGD(model.parameters(), lr=0.01)

optimizer\_adadelta = optim.Adadelta(model.parameters(), lr=0.0000001)

optimizer\_nag = optim.SGD(model.parameters(), lr=0.0000001, momentum=0.9, nesterov=True)

optimizer\_adam = optim.Adam(model.parameters(), lr=0.0000001)

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}")

    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()

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),

    "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()

        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()

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)

    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()

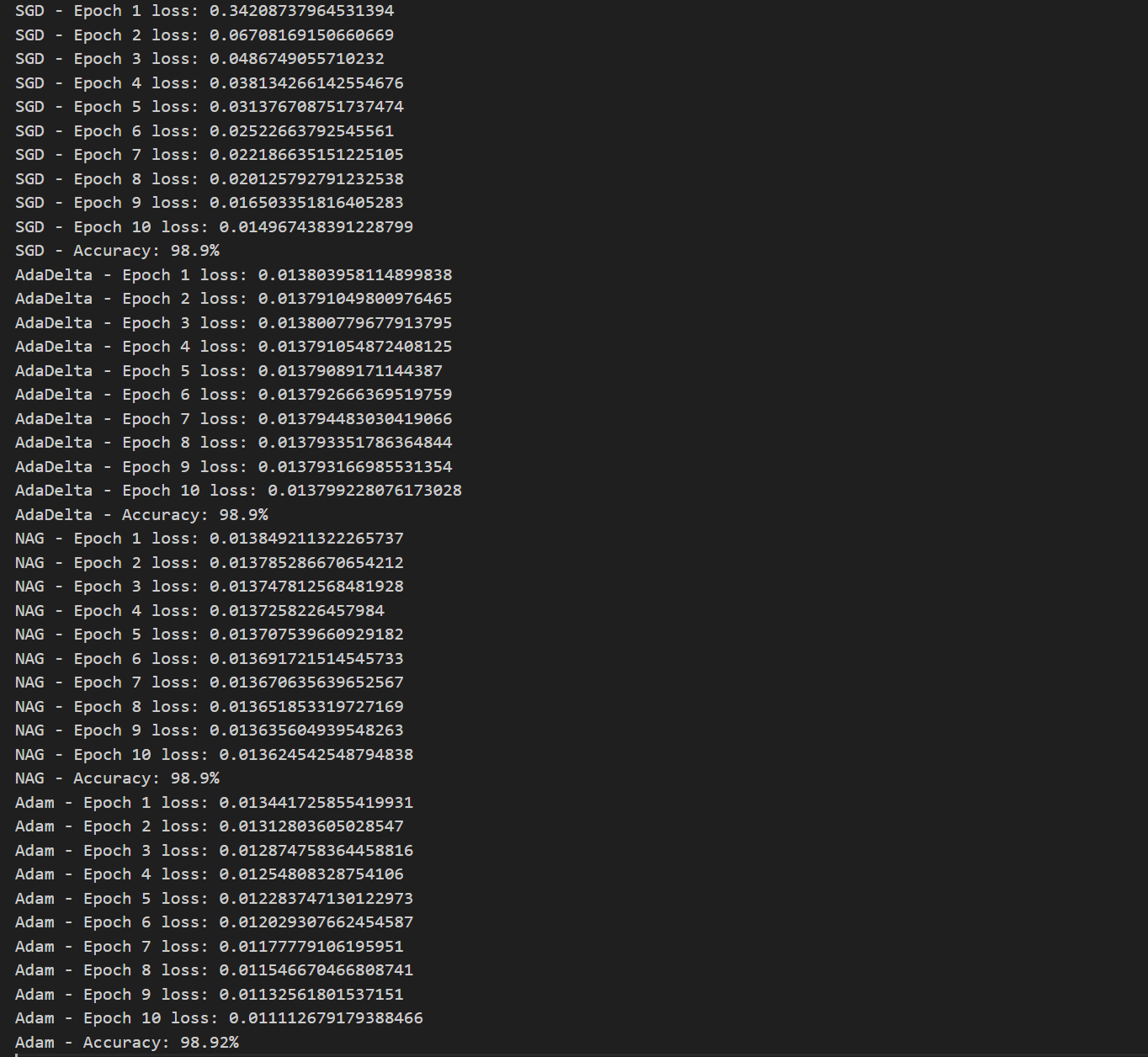
    plt.show()

print("Accuracies for different optimizers:")

for optimizer\_name, accuracy in accuracies.items():

    print(f"{optimizer\_name}: {accuracy:.2f}%")

# Результаты



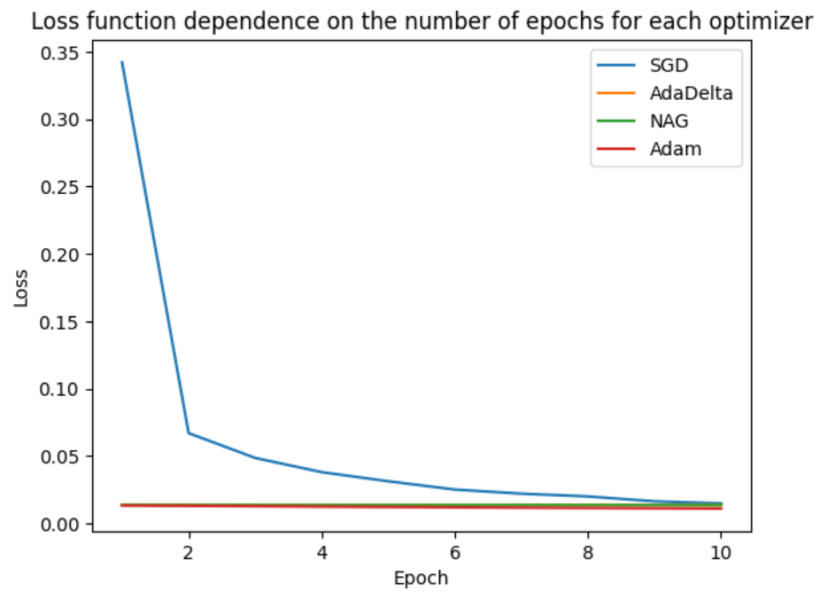


Рисунок - Результат работы LeNet с оптимизаторами SGD, AdaDelta, NAG, Adam

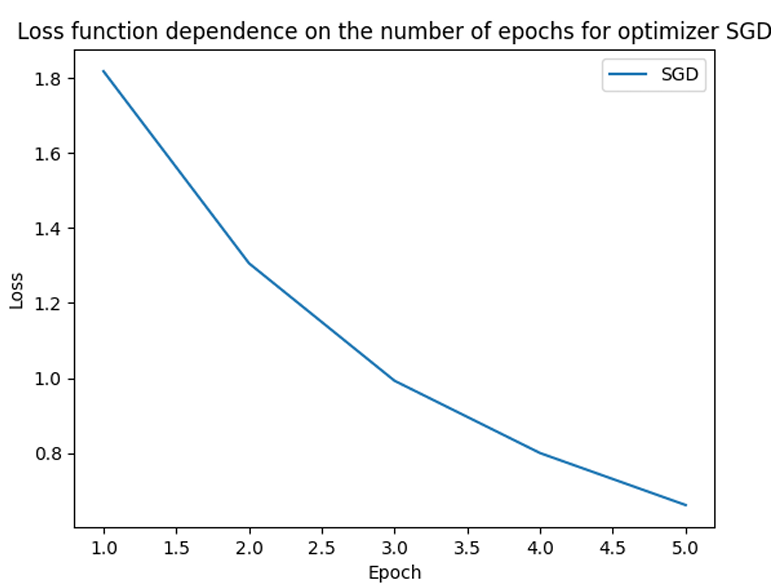


Рисунок - Результат работы VGG16 с оптимизатором SGD (точность: 75.78%)

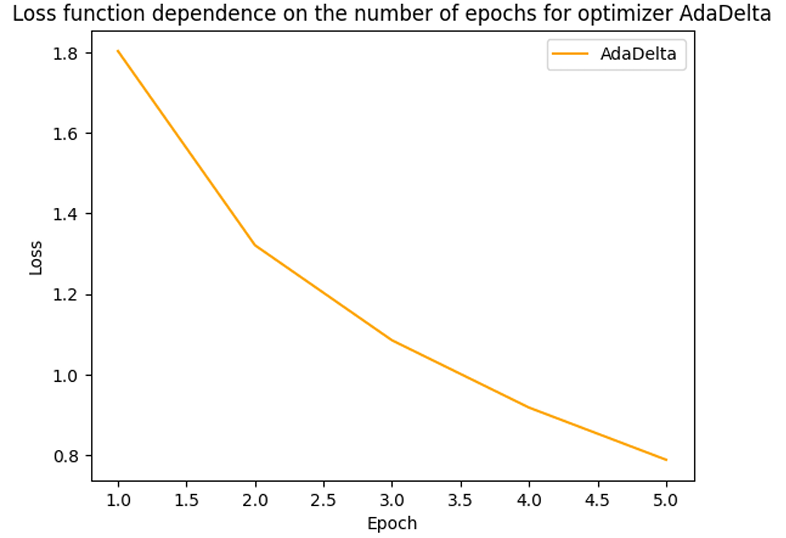


Рисунок - Результат работы VGG16 с оптимизатором AdaDelta (точность: 70.1%)

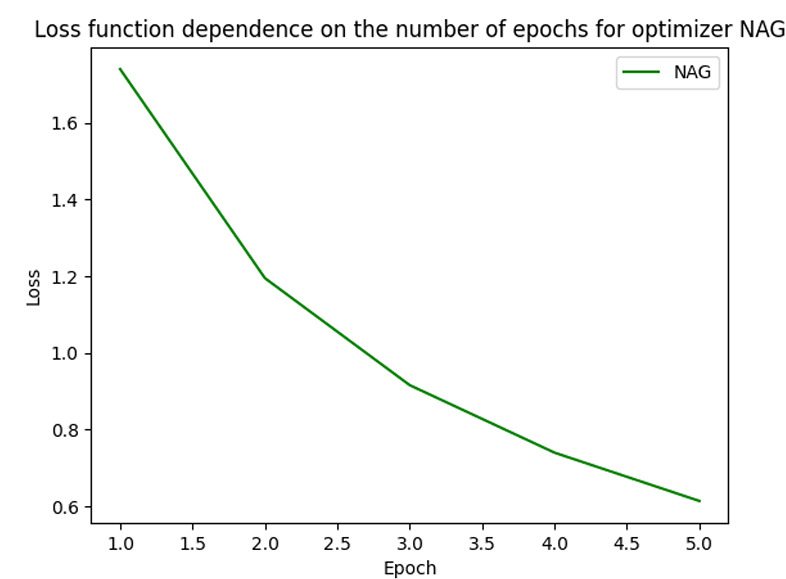


Рисунок - Результат работы VGG16 с оптимизатором NAG (точность: 77.75%)

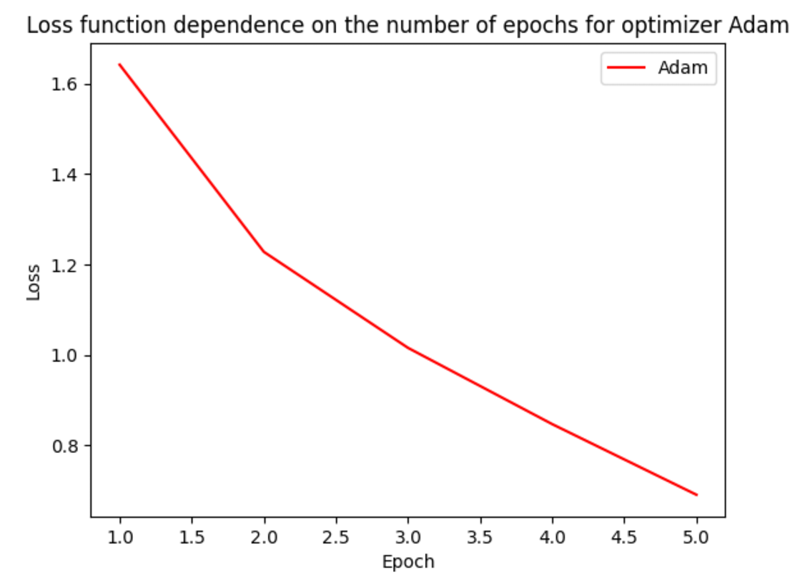


Рисунок - Результат работы VGG16 с оптимизатором Adam (точность: 67.46%)

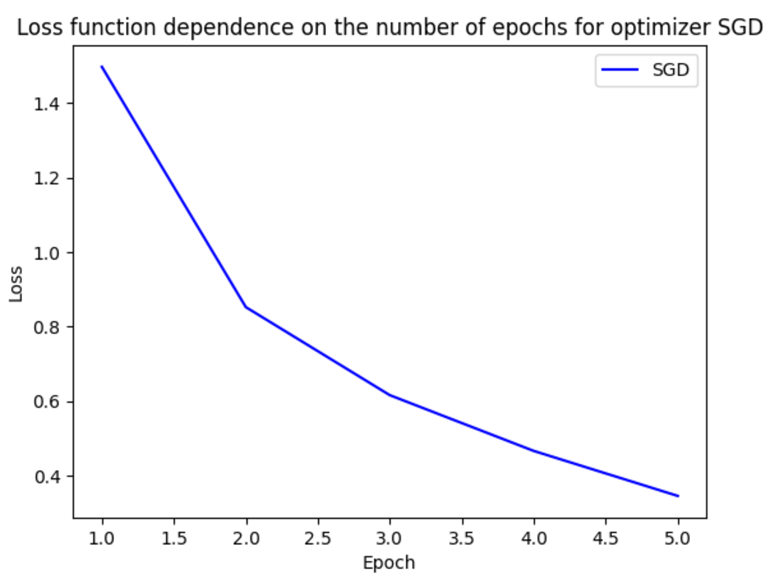


Рисунок - Результат работы ResNet (34) с оптимизатором SGD (точность: 74.22%)

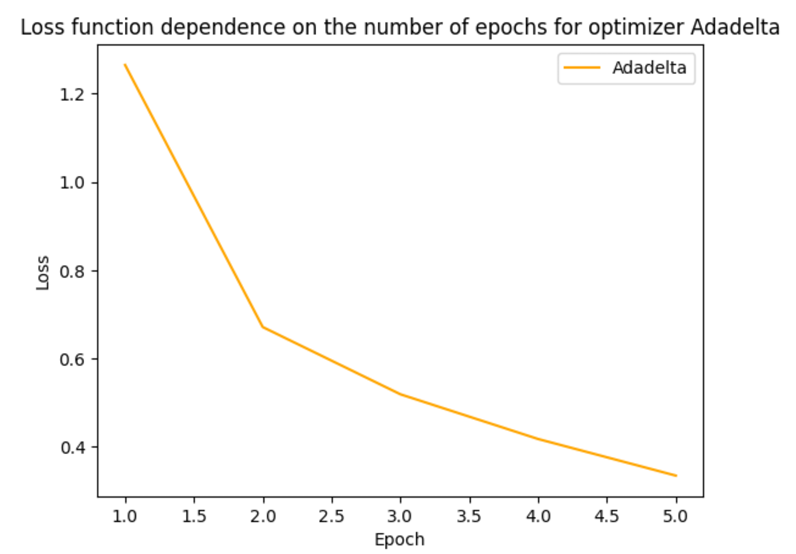


Рисунок - Результат работы ResNet (34) с оптимизатором AdaDelta (точность: 68.96%)

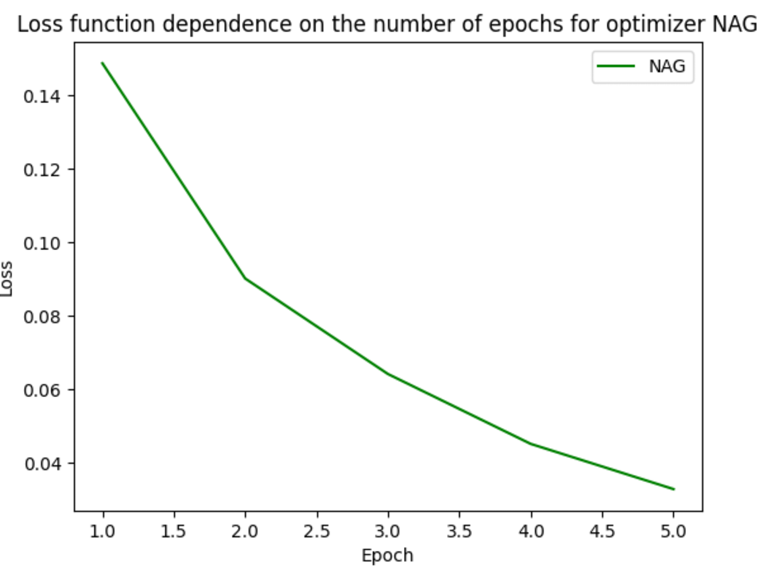


Рисунок - Результат работы ResNet (34) с оптимизатором NAG (точность: 87.34%)

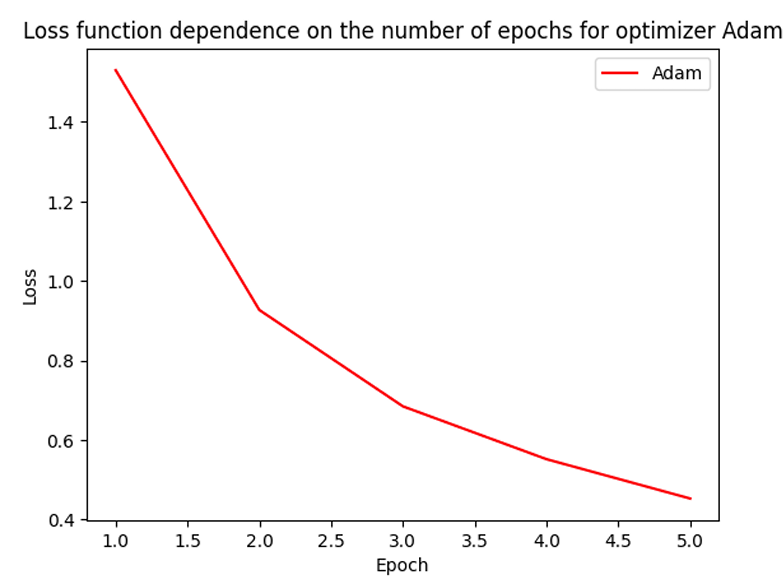


Рисунок - Результат работы ResNet (34) с оптимизатором Adam (точность: 80.01%)

# Вывод

В ходе выполнения данной работы были разработаны нейронные сети LeNet, VGG16, ResNet (34) с оптимизаторами SGD, AdaDelta, NAG, Adam. По результатам для LeNet лучшим методом оптимизации является Adam, для VGG16 и ResNet (34) – NAG.