



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»**  
**«Свёрточные нейронные сети»**

Студент группы ИУ9-71Б Афанасьев И.

Преподаватель Каганов Ю.Т.

*Москва 2024*

# 1 Цель работы

1. Изучение основных архитектур свёрточных нейронных сетей: LeNet-5, VGG-16, ResNet-34.
2. Обучение свёрточных сетей с использованием различных оптимизаторов и фреймворка PyTorch.

# 2 Реализация

В листинге 1 приводится исходный код программы на языке Python с использованием фреймворка PyTorch.

Листинг 1: Файл hw5.py

```
1 import datetime
2 import torch
3 import torch.nn as nn
4 import torch.optim as optim
5
6 from torchvision import datasets, transforms, models
7
8 DATA_PATH = '../..'/datasets/'
9 BATCH_SIZE = 100
10 MOMENTUM = 0.9
11 EPOCHS = 20
12
13
14 class LeNet5(nn.Module):
15     def __init__(self, num_classes):
16         super(LeNet5, self).__init__()
17         self.layer1 = nn.Sequential(
18             nn.Conv2d(1, 6, kernel_size=5, stride=1, padding=0),
19             nn.BatchNorm2d(6),
20             nn.ReLU(),
21             nn.MaxPool2d(kernel_size=2, stride=2))
22         self.layer2 = nn.Sequential(
23             nn.Conv2d(6, 16, kernel_size=5, stride=1, padding=0),
24             nn.BatchNorm2d(16),
```

```

25         nn.ReLU(),
26         nn.MaxPool2d(kernel_size=2, stride=2))
27     self.fc = nn.Linear(400, 120)
28     self.relu = nn.ReLU()
29     self.fc1 = nn.Linear(120, 84)
30     self.relu1 = nn.ReLU()
31     self.fc2 = nn.Linear(84, num_classes)
32
33     def forward(self, x):
34         out = self.layer1(x)
35         out = self.layer2(out)
36         out = out.reshape(out.size(0), -1)
37         out = self.fc(out)
38         out = self.relu(out)
39         out = self.fc1(out)
40         out = self.relu1(out)
41         out = self.fc2(out)
42         return out
43
44
45     def train(n_epochs, optimizer, model, loss_fn, train_loader):
46         for epoch in range(1, n_epochs + 1):
47             loss_train = 0.0
48             for imgs, labels in train_loader:
49                 imgs = imgs.to(device=device)
50                 labels = labels.to(device=device)
51                 outputs = model(imgs)
52                 loss = loss_fn(outputs, labels)
53
54                 optimizer.zero_grad()
55                 loss.backward()
56                 optimizer.step()
57
58             loss_train += loss.item()
59
60         print('{} Epoch {}, Training loss {}'.format(
61             datetime.datetime.now(), epoch,
62             loss_train / len(train_loader)))
63
64

```

```

65 def calculate_accuracy(model, train_loader, test_loader):
66     accdict = {}
67     for name, loader in [("train", train_loader), ("test", test_loader)]:
68         correct = 0
69         total = 0
70
71         with torch.no_grad():
72             for imgs, labels in loader:
73                 imgs = imgs.to(device=device)
74                 labels = labels.to(device=device)
75                 outputs = model(imgs)
76                 _, predicted = torch.max(outputs, dim=1)
77                 total += labels.shape[0]
78                 correct += int((predicted == labels).sum())
79
80         print("Accuracy {}: {:.3f}".format(name, correct / total))
81         accdict[name] = correct / total
82     return accdict
83
84
85 if __name__ == '__main__':
86     device = (torch.device('cuda') if torch.cuda.is_available()
87              else torch.device('cpu'))
88     print(f'Using {device}')
89
90     mnist_train = datasets.MNIST(
91         DATA_PATH, train=True, download=True, transform=transforms.Compose([
92             transforms.Resize((32, 32)),
93             transforms.ToTensor(),
94             transforms.Normalize(mean=(0.1307,), std=(0.3081,))]))
95     mnist_test = datasets.MNIST(
96         DATA_PATH, train=False, download=True, transform=transforms.Compose([
97             transforms.Resize((32, 32)),
98             transforms.ToTensor(),
99             transforms.Normalize(mean=(0.1325,), std=(0.3105,))]))
100
101     cifar10_train = datasets.CIFAR10(
102         DATA_PATH, train=True, download=True, transform=transforms.Compose([
103             transforms.ToTensor(),
104             transforms.Normalize((0.4915, 0.4823, 0.4468),

```

```

105         (0.2470, 0.2435, 0.2616))
106     )))
107     cifar10_test = datasets.CIFAR10(
108         DATA_PATH, train=False, download=True, transform=transforms.Compose([
109             transforms.ToTensor(),
110             transforms.Normalize((0.4915, 0.4823, 0.4468),
111                                 (0.2470, 0.2435, 0.2616))
112         ]))
113
114     loss_fn = nn.CrossEntropyLoss()
115
116     print('LeNet5, MNIST')
117     model = LeNet5(num_classes=10).to(device=device)
118     train_loader = torch.utils.data.DataLoader(
119         mnist_train, batch_size=BATCH_SIZE, shuffle=True)
120     test_loader = torch.utils.data.DataLoader(
121         mnist_test, batch_size=BATCH_SIZE, shuffle=True)
122
123     print('SGD')
124     optimizer = optim.SGD(model.parameters(), lr=1e-2)
125     train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
126          loss_fn=loss_fn, train_loader=train_loader)
127     calculate_accuracy(model, train_loader, test_loader)
128
129     print('Adadelata')
130     optimizer = optim.Adadelata(model.parameters(), lr=1e-2)
131     train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
132          loss_fn=loss_fn, train_loader=train_loader)
133     calculate_accuracy(model, train_loader, test_loader)
134
135     print('NAG')
136     optimizer = optim.SGD(model.parameters(), lr=1e-2,
137                            momentum=MOMENTUM, nesterov=True)
138     train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
139          loss_fn=loss_fn, train_loader=train_loader)
140     calculate_accuracy(model, train_loader, test_loader)
141
142     print('Adam')
143     optimizer = optim.Adam(model.parameters(), lr=1e-2)
144     train(n_epochs=EPOCHS, optimizer=optimizer, model=model,

```

```

145         loss_fn=loss_fn, train_loader=train_loader)
146 calculate_accuracy(model, train_loader, test_loader)
147
148 print('VGG16, CIFAR10')
149 model = models.vgg16(num_classes=10, dropout=0.5).to(device=device)
150 train_loader = torch.utils.data.DataLoader(
151     cifar10_train, batch_size=BATCH_SIZE, shuffle=True)
152 test_loader = torch.utils.data.DataLoader(
153     cifar10_test, batch_size=BATCH_SIZE, shuffle=True)
154
155 print('SGD')
156 optimizer = optim.SGD(model.parameters(), lr=1e-2)
157 train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
158     loss_fn=loss_fn, train_loader=train_loader)
159 calculate_accuracy(model, train_loader, test_loader)
160
161 print('Adadelta')
162 optimizer = optim.Adadelta(model.parameters(), lr=1e-2)
163 train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
164     loss_fn=loss_fn, train_loader=train_loader)
165 calculate_accuracy(model, train_loader, test_loader)
166
167 print('NAG')
168 optimizer = optim.SGD(model.parameters(), lr=1e-2,
169     momentum=MOMENTUM, nesterov=True)
170 train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
171     loss_fn=loss_fn, train_loader=train_loader)
172 calculate_accuracy(model, train_loader, test_loader)
173
174 print('Adam')
175 optimizer = optim.Adam(model.parameters(), lr=1e-2)
176 train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
177     loss_fn=loss_fn, train_loader=train_loader)
178 calculate_accuracy(model, train_loader, test_loader)
179
180 print('ResNet34, CIFAR10')
181 model = models.resnet34(num_classes=10).to(device)
182
183 print('SGD')
184 optimizer = optim.SGD(model.parameters(), lr=1e-2)

```

```

185     train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
186           loss_fn=loss_fn, train_loader=train_loader)
187     calculate_accuracy(model, train_loader, test_loader)
188
189     print('Adadelta')
190     optimizer = optim.Adadelta(model.parameters(), lr=1e-2)
191     train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
192           loss_fn=loss_fn, train_loader=train_loader)
193     calculate_accuracy(model, train_loader, test_loader)
194
195     print('NAG')
196     optimizer = optim.SGD(model.parameters(), lr=1e-2,
197                           momentum=MOMENTUM, nesterov=True)
198     train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
199           loss_fn=loss_fn, train_loader=train_loader)
200     calculate_accuracy(model, train_loader, test_loader)
201
202     print('Adam')
203     optimizer = optim.Adam(model.parameters(), lr=1e-2)
204     train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
205           loss_fn=loss_fn, train_loader=train_loader)
206     calculate_accuracy(model, train_loader, test_loader)

```

## 3 Результаты сравнения

### 3.1 Архитектура LeNet-5

Для обучения нейронной сети с архитектурой LeNet-5 используется датасет MNIST. В таблице 1 приводятся результаты обучения. Точность измеряется на тестовых данных

### 3.2 Архитектура VGG-16

Для обучения нейронной сети с архитектурой VGG-16 используется датасет CIFAR-10. В таблице 2 приводятся результаты обучения.

Оптимизатор	Эпохи	Коэффициент обучения	Точность, %
SGD	20	$10^{-2}$	98.7
AdaDelta	20	$10^{-2}$	98.8
NAD	20	$10^{-2}$	99.1
Adam	20	$10^{-2}$	98.9

Таблица 1: Вариация гиперпараметров LeNet-5.

Оптимизатор	Эпохи	Коэффициент обучения	Dropout	Точность, %
SGD	20	$10^{-2}$	0.5	73.9
AdaDelta	20	$10^{-2}$	0.5	75.9
NAD	20	$10^{-2}$	0.5	80.9
Adam	20	$10^{-2}$	0.5	10.0

Таблица 2: Вариация гиперпараметров VGG-16.

По таблице 2 видно, что с использованием оптимизатора Adam достигается очень низкая точность — идёт застревание в локальном минимуме. Изменением гиперпараметров (количество эпох, коэффициент обучения, dropout) решить проблему не удаётся.

### 3.3 Архитектура ResNet-34

Для обучения нейронной сети с архитектурой ResNet-34 используется датасет CIFAR-10. В таблице 3 приводятся результаты обучения.

Оптимизатор	Эпохи	Коэффициент обучения	Dropout	Точность, %
SGD	40	$10^{-2}$	0.5	65.4
AdaDelta	40	$10^{-2}$	0.5	66.3
NAD	40	$10^{-2}$	0.5	75.2
Adam	20	$10^{-2}$	0.5	74.8

Таблица 3: Вариация гиперпараметров ResNet-34.