

Министерство образования Республики Беларусь
Учреждение образования
«Брестский Государственный технический университет»
Кафедра ИИТ

Лабораторная работа №1
По дисциплине «ОИвИС»
Тема: “Обучение классификаторов средствами библиотеки
PyTorch”

Выполнил:
Студент 4 курса
Группы ИИ-23
Лапин В. А.
Проверила:
Андренко К.В.

Цель: научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.

Вариант 8.

Выборка: CIFAR-10. Размер исходного изображения: 32*32 Оптимизатор: Adam.

1. Выполнить конструирование своей модели СНС, обучить ее на выборке по заданию (использовать torchvision.datasets). Предпочтение отдавать как можно более простым архитектурам, базирующимся на базовых типах слоев (сверточный, полносвязный, подвыборочный, слой нелинейного преобразования). Оценить эффективность обучения на тестовой выборке, построить график изменения ошибки (matplotlib);
2. Ознакомьтесь с state-of-the-art результатами для предлагаемых выборок (<https://paperswithcode.com/task/image-classification>). Сделать выводы о результатах обучения СНС из п. 1;
3. Реализовать визуализацию работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата);
4. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

Код программы:

```
from torchvision import transforms
import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np

train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.RandomResizedCrop(32, scale=(0.8, 1.0)),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

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

train_loader = torch.utils.data.DataLoader(
    torchvision.datasets.CIFAR10(data_dir, train=True, download=True,
                                transform=train_transform),
    batch_size=64, shuffle=True)

test_loader = torch.utils.data.DataLoader(
    torchvision.datasets.CIFAR10(data_dir, train=False, download=True,
                                transform=test_transform),
    batch_size=64, shuffle=False)

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

        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)

        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(128 * 4 * 4, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)

        self.relu = nn.ReLU()
```

```

self.dropout = nn.Dropout(0.3)

def forward(self, x):
    x = self.pool(self.relu(self.conv1(x)))
    x = self.pool(self.relu(self.conv2(x)))
    x = self.pool(self.relu(self.conv3(x)))

    x = x.view(-1, 128 * 4 * 4)

    x = self.dropout(self.relu(self.fc1(x)))
    x = self.dropout(self.relu(self.fc2(x)))
    x = self.fc3(x)

    return x

model = CNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

def train(model, loader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in 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()

        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    accuracy = 100 * correct / total
    return running_loss / len(loader), accuracy

def test(model, loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

```

```

with torch.no_grad():
    for images, labels in loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        running_loss += loss.item()

    _, predicted = torch.max(outputs, 1)
    correct += (predicted == labels).sum().item()
    total += labels.size(0)

accuracy = 100 * correct / total
return running_loss / len(loader), accuracy

device = torch.device('mps' if torch.mps.is_available() else 'cpu')
model = model.to(device)

train_losses = []
test_losses = []
train_accuracies = []
test_accuracies = []

num_epochs = 15

for epoch in range(num_epochs):
    train_loss, train_accuracy = train(model, train_loader, criterion, optimizer, device)
    test_loss, test_accuracy = test(model, test_loader, criterion, device)

    train_losses.append(train_loss)
    test_losses.append(test_loss)
    train_accuracies.append(train_accuracy)
    test_accuracies.append(test_accuracy)

    print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.2f}%',
          f'Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.2f}%')

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Train Loss')
plt.plot(test_losses, label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Test Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(test_accuracies, label='Test Accuracy')

```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.title('Training and Test Accuracy')
plt.legend()
```

```
plt.tight_layout()
plt.show()
```

```
def imshow(img):
    img = img / 2 + 0.5
    np_img = img.numpy()
    plt.imshow(np.transpose(np_img, (1, 2, 0)))
    plt.axis('off')
    plt.show()
```

```
def test_random_image(model, loader, device):
    model.eval()
    images, labels = next(iter(loader))
    images, labels = images.to(device), labels.to(device)
```

```
    import random
    index = random.randint(0, images.size(0) - 1)
    image = images[index].unsqueeze(0)
    label = labels[index].item()
```

```
    output = model(image)
    _, predicted = torch.max(output, 1)
    predicted = predicted.item()
```

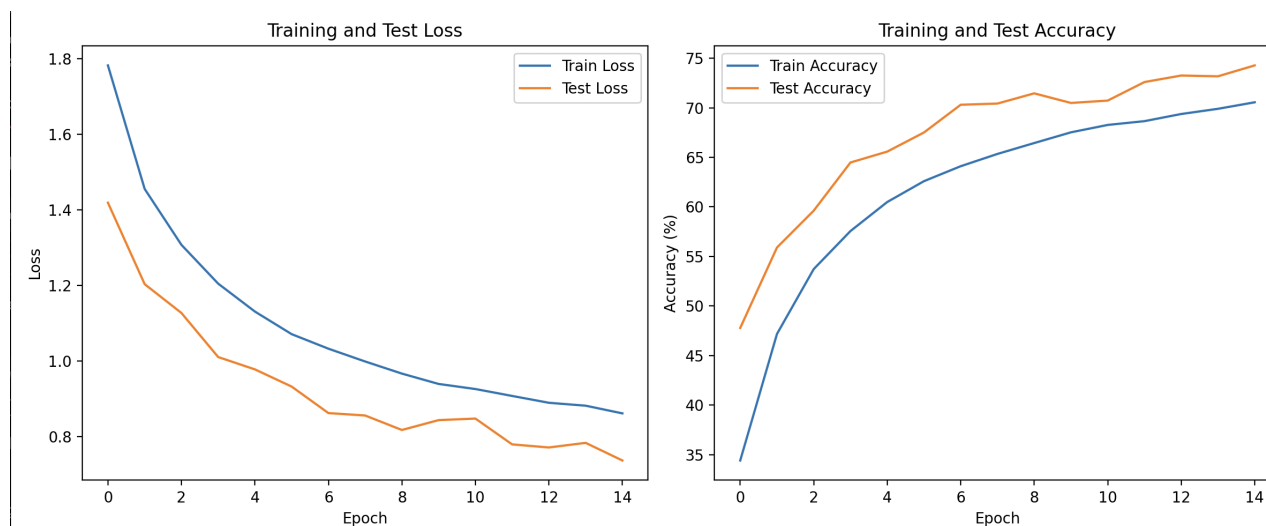
```
    imshow(image.cpu().squeeze())
    print(f'Predicted: {predicted}, Actual: {label}')
```

```
test_random_image(model, test_loader, device)
```

1. Результат работы программы:

```
Epoch 8/15, Train Loss: 0.9993, Train Accuracy: 65.35%, Test Loss: 0.8564, Test Accuracy: 70.43%
Epoch 9/15, Train Loss: 0.9673, Train Accuracy: 66.44%, Test Loss: 0.8181, Test Accuracy: 71.46%
Epoch 10/15, Train Loss: 0.9399, Train Accuracy: 67.53%, Test Loss: 0.8442, Test Accuracy: 70.49%
Epoch 11/15, Train Loss: 0.9265, Train Accuracy: 68.27%, Test Loss: 0.8483, Test Accuracy: 70.73%
Epoch 12/15, Train Loss: 0.9081, Train Accuracy: 68.65%, Test Loss: 0.7799, Test Accuracy: 72.60%
Epoch 13/15, Train Loss: 0.8901, Train Accuracy: 69.38%, Test Loss: 0.7718, Test Accuracy: 73.26%
Epoch 14/15, Train Loss: 0.8824, Train Accuracy: 69.90%, Test Loss: 0.7840, Test Accuracy: 73.18%
Epoch 15/15, Train Loss: 0.8620, Train Accuracy: 70.56%, Test Loss: 0.7373, Test Accuracy: 74.28%
```

График изменения ошибок:



2. SOTA-результаты для выборки:

Модель ConvMLP-S является гибридной архитектурой, которая сочетает в себе эффективность сверточных слоев (CNN) для извлечения локальных признаков со способностью многослойных перцептронов (MLP) моделировать глобальные зависимости. Гибридная архитектура обеспечивает взаимодействие между локальными и глобальными признаками. Сверточные слои извлекают низкоуровневые признаки, а MLP-блоки моделируют сложные взаимосвязи между ними. Архитектура оптимизирована для эффективного использования вычислительных ресурсов. Вычисления рационально распределяются между CNN и MLP компонентами модели.

Результат ConvMLP-S для выборки:

Model Name ↑↓	Percentage correct ↑
ConvMLP-S	98

Ссылка на статью: <https://hyper.ai/en/papers/2109.04454>

Результат сверточной сети с 6 слоями с обучаемыми параметрами за 15 эпох:

```
Epoch 8/15, Train Loss: 0.9993, Train Accuracy: 65.35%, Test Loss: 0.8564, Test Accuracy: 70.43%
Epoch 9/15, Train Loss: 0.9673, Train Accuracy: 66.44%, Test Loss: 0.8181, Test Accuracy: 71.46%
Epoch 10/15, Train Loss: 0.9399, Train Accuracy: 67.53%, Test Loss: 0.8442, Test Accuracy: 70.49%
Epoch 11/15, Train Loss: 0.9265, Train Accuracy: 68.27%, Test Loss: 0.8483, Test Accuracy: 70.73%
Epoch 12/15, Train Loss: 0.9081, Train Accuracy: 68.65%, Test Loss: 0.7799, Test Accuracy: 72.60%
Epoch 13/15, Train Loss: 0.8901, Train Accuracy: 69.38%, Test Loss: 0.7718, Test Accuracy: 73.26%
Epoch 14/15, Train Loss: 0.8824, Train Accuracy: 69.90%, Test Loss: 0.7840, Test Accuracy: 73.18%
Epoch 15/15, Train Loss: 0.8620, Train Accuracy: 70.56%, Test Loss: 0.7373, Test Accuracy: 74.28%
```


Разница в точности обусловлена тем, что CNN представляет собой компактную 6-слойную сеть с примерно 500 тысячами параметров, тогда как convMLP-S — это значительно более глубокая и сложная архитектура с миллионами параметров и продвинутыми архитектурными решениями:

1. Принципиально разной емкостью моделей - convMLP-S имеет значительно больше параметров и слоев.
2. Архитектурными преимуществами специализированной сети - batch normalization в convMLP-S.
3. Оптимизированной под датасет архитектурой - convMLP-S проектировалась специально для задач типа CIFAR-10.
4. Продвинутыми техниками аугментации данных - более сложные преобразования входных изображений.
5. Тщательным подбором гиперпараметров - оптимизированные learning rate, вес decay и другие параметры.
6. Более глубокой архитектурой - возможность извлекать более сложные иерархические признаки.
7. Использованием механизмов внимания - convMLP-S может включать attention blocks для фокусировки на важных участках изображения.
8. Оптимизированной картой признаков - специально подобранная карта каналов (channel dimensions) для максимальной эффективности.

3. Визуализация работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата).

Predicted: 1, Actual: 1



Вывод: обучил классификатор средствами библиотеки PyTorch.