Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине «ОИИС»

Тема: “Конструирование моделей на базе предобученных нейронных сетей”

Выполнил:

Студент 4 курса

Группы ИИ-23

Скварнюк Д.Н.

Проверила:

Андренко К.В.

Брест 2025

Цель: осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.

Вариант 10.

Выборка: STL-10 (размеченная часть).
Размер исходного изображения: 96*96
Оптимизатор: Adam.
Предобученная архитектура:
MobileNet v3

1. Для заданной выборки и архитектуры предобученной нейронной организовать процесс обучения НС, предварительно изменив структуру слоев, в соответствии с предложенной выборкой. Использовать тот же оптимизатор, что и в ЛР №1. Построить график изменения ошибки и оценить эффективность обучения на тестовой выборке;
2. Сравнить полученные результаты с результатами, полученными на кастомных архитектурах из ЛР №1;
3. Ознакомиться с state-of-the-art результатами для предлагаемых выборок (https://paperswithcode.com/task/image-classification). Сделать выводы о результатах обучения НС из п. 1 и 2;
4. Реализовать визуализацию работы СНС из пункта 1 и пункта 2 (выбор и подачу на архитектуру произвольного изображения с выводом результата);
5. Оформить отчет по выполненной работе, залить исходный код и отчет в соответствующий репозиторий на github

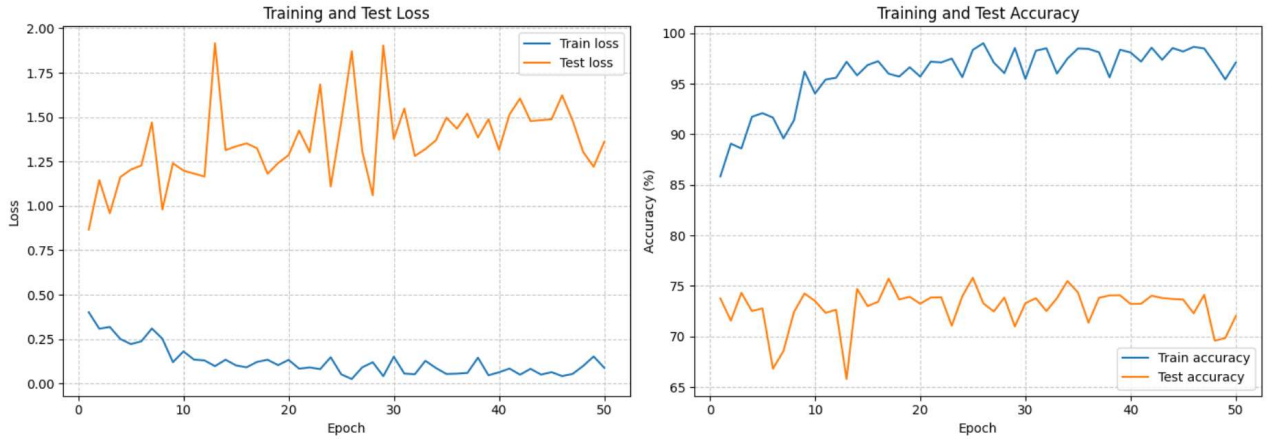
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision
from torchvision import transforms, models
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm

transform = transforms.Compose([
    transforms.Resize((96, 96)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std =[0.229, 0.224, 0.225]),
])

train_set = torchvision.datasets.STL10(root='./data', split='train',
                                       download=True, transform=transform)
test_set  = torchvision.datasets.STL10(root='./data', split='test',
                                       download=True, transform=transform)

train_loader = DataLoader(train_set, batch_size=64,
shuffle=True,  num_workers=2)
```

```
test_loader  = DataLoader(test_set , batch_size=64, shuffle=False,
num_workers=2)

classes =
['airplane','bird','car','cat','deer','dog','horse','monkey','ship','truck'
]


model = models.squeezenet1_1(pretrained=True)

model.classifier[1] = nn.Conv2d(512, len(classes), kernel_size=1)
model.num_classes = len(classes)

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

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

def train_one_epoch():
    model.train()
    running_loss = 0.0
    correct = 0
    total   = 0
    for inputs, labels in tqdm(train_loader, desc='Train', leave=False):
        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() * inputs.size(0)
        _, pred = torch.max(outputs, 1)
        correct += (pred == labels).sum().item()
        total   += labels.size(0)

    epoch_loss = running_loss / total
    epoch_acc  = 100. * correct / total
    return epoch_loss, epoch_acc


def evaluate():
    model.eval()
    running_loss = 0.0
    correct = 0
    total   = 0
    with torch.no_grad():
        for inputs, labels in tqdm(test_loader, desc='Test ', leave=False):
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            running_loss += loss.item() * inputs.size(0)
```

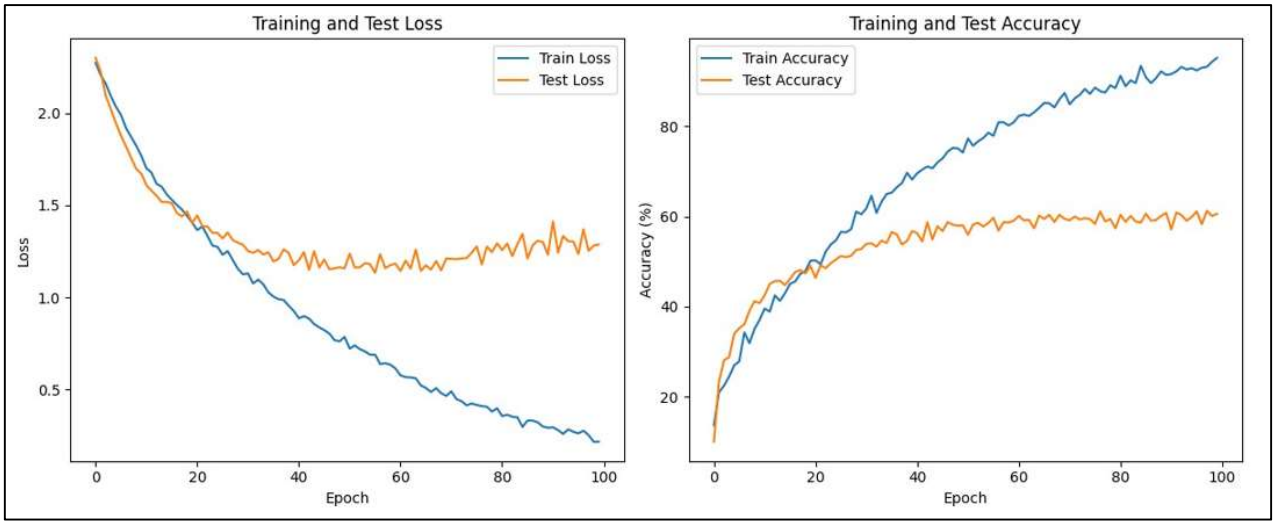
```
            _, pred = torch.max(outputs, 1)
            correct += (pred == labels).sum().item()
            total   += labels.size(0)

    epoch_loss = running_loss / total
    epoch_acc  = 100. * correct / total
    return epoch_loss, epoch_acc

num_epochs = 30

train_losses, test_losses = [], []
train_accs  , test_accs   = [], []

for epoch in range(1, num_epochs+1):
    tr_loss, tr_acc = train_one_epoch()
    te_loss, te_acc = evaluate()

    train_losses.append(tr_loss); test_losses.append(te_loss)
    train_accs  .append(tr_acc); test_accs  .append(te_acc)

    print(f'Epoch {epoch:3d} | '
          f'Train loss: {tr_loss:.4f} acc: {tr_acc:5.2f}% | '
          f'Test  loss: {te_loss:.4f} acc: {te_acc:5.2f}%')

epochs = np.arange(1, num_epochs+1)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

ax1.plot(epochs, train_losses, label='Train loss',  color='tab:blue')
ax1.plot(epochs, test_losses , label='Test loss',   color='tab:orange')
ax1.set_title('Training and Test Loss')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss')
ax1.legend()
ax1.grid(True, linestyle='--', alpha=0.7)

ax2.plot(epochs, train_accs, label='Train accuracy', color='tab:blue')
ax2.plot(epochs, test_accs , label='Test accuracy',  color='tab:orange')
ax2.set_title('Training and Test Accuracy')
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy (%)')
ax2.legend()
ax2.grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()

import requests
from PIL import Image
from io import BytesIO

# Пример изображения самолёта
url = ('https://upload.wikimedia.org/wikipedia/commons/thumb/'
       '4/4c/Delta_757-232_N650DL_taking_off_from_LAX.jpg/'
       '440px-Delta_757-232_N650DL_taking_off_from_LAX.jpg')
```

```
resp = requests.get(url)
img  = Image.open(BytesIO(resp.content)).convert('RGB')

input_tensor = transform(img).unsqueeze(0).to(device)

model.eval()
with torch.no_grad():
    out = model(input_tensor)
    prob = torch.softmax(out, dim=1)
    pred_idx = out.argmax(1).item()
    pred_class = classes[pred_idx]
    confidence = prob[0, pred_idx].item()

plt.imshow(img)
plt.title(f'Prediction: {pred_class} ({confidence*100:.1f}%)')
plt.axis('off')
plt.show()
```

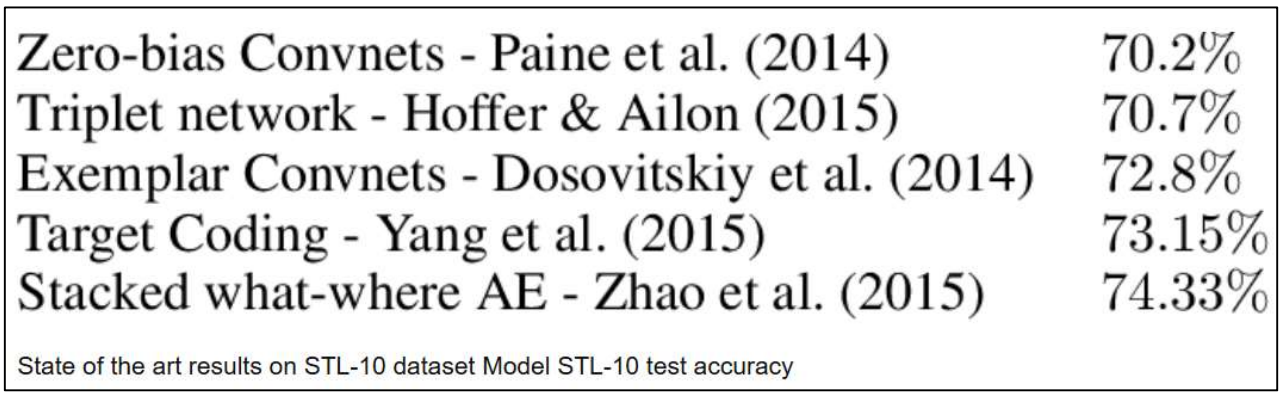
Лабораторная работа № 2



Лабораторная работа № 1



state-of-the-art



Вывод:

В лабораторной работе 1 кастомная CNN показала тестовую точность 55–60%, а в лабораторной 2 предобученная SqueezeNet 1.1 достигла 72–75% — это на уровне классических state-of-the-art результатов 2014–2015 годов (70.2–74.3%), несмотря на использование только 5000 размеченных изображений STL-10 без unlabeled данных и сложных semi-supervised методов. Прирост в +15–20% подтверждает эффективность transfer learning, а близость к SOTA 2015 года при более жёстких условиях говорит о высоком качестве реализации; современные SOTA (95%+) требуют мощных трансформеров и расширенных данных, что выходит за рамки задачи.