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

Лабораторная работа №2
По дисциплине «ОИвИС»
Тема: “Конструирование моделей на базе предобученных нейронных сетей”

Выполнил:
Студент 4 курса
Группы ИИ-23
Глухарев Д.Е.
Проверила:
Андренко К.В.

Брест 2025

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

Вариант 5.

Выборка: STL-10 (размеченная часть).
Размер исходного изображения: 96*96
Оптимизатор: SGD.

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.

Ход работы:

```
import argparse
import os
from tqdm import tqdm
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms, models


CLASS_NAMES = ('airplane','bird','car','cat','deer','dog','horse','monkey','ship','truck')


# === Предобученная DenseNet121 ===
class DenseNet121Modified(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        # Загружаем предобученную DenseNet121
        self.model = models.densenet121(weights=models.DenseNet121_Weights.IMAGENET1K_V1)
        # Заменяем последний классификатор под нашу задачу
        in_features = self.model.classifier.in_features
        self.model.classifier = nn.Linear(in_features, num_classes)

    def forward(self, x):
        return self.model(x)
```

```
class STL10Dataset(Dataset):
    def __init__(self, data_file, labels_file, transform=None):
        self.transform = transform
        with open(data_file, 'rb') as f:
            data = np.fromfile(f, dtype=np.uint8)
            data = data.reshape(-1, 3, 96, 96)
            self.data = np.transpose(data, (0, 2, 3, 1))
        with open(labels_file, 'rb') as f:
            self.labels = np.fromfile(f, dtype=np.uint8) - 1

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        img, label = self.data[idx], self.labels[idx]
        img = Image.fromarray(img)
        if self.transform:
            img = self.transform(img)
        return img, label



def get_dataloaders(batch_size=64, num_workers=2):
    transform_train = transforms.Compose([
        transforms.Resize((96,96)),
        transforms.RandomHorizontalFlip(),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])  # важно для предобученной сети
    ])
    transform_test = 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 = STL10Dataset(
        data_file='./data/train_X.bin',
        labels_file='./data/train_y.bin',
        transform=transform_train
    )
    test_set = STL10Dataset(
        data_file='./data/test_X.bin',
        labels_file='./data/test_y.bin',
        transform=transform_test
    )
```

```
    train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=num_workers)
    test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=False, num_workers=num_workers)
    return train_loader, test_loader



def train_one_epoch(model, device, dataloader, criterion, optimizer):
    model.train()
    running_loss = 0.0
    for images, targets in tqdm(dataloader, desc='Train batches', leave=False):
        images = images.to(device)
        targets = targets.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)
    return running_loss / len(dataloader.dataset)



def evaluate(model, device, dataloader, criterion=None):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for images, targets in dataloader:
            images = images.to(device)
            targets = targets.to(device)
            outputs = model(images)
            if criterion is not None:
                loss = criterion(outputs, targets)
                running_loss += loss.item() * images.size(0)
            preds = outputs.argmax(dim=1)
            correct += (preds == targets).sum().item()
            total += targets.size(0)
    avg_loss = running_loss / len(dataloader.dataset) if criterion is not None else None
    accuracy = correct / total
    return avg_loss, accuracy



def plot_losses(train_losses, val_losses, out_path=None):
    epochs = np.arange(1, len(train_losses)+1)
    plt.figure(figsize=(8,5))
    plt.plot(epochs, train_losses, label='Train loss')
    plt.plot(epochs, val_losses, label='Val loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
```

```
    plt.title('Loss curve')
    plt.legend()
    plt.grid(True)
    if out_path:
        plt.savefig(out_path, dpi=150)
        print(f'Saved loss plot to {out_path}')
    plt.show()


def show_predictions(model, device, dataloader, num_images=8):
    model.eval()
    images_shown = 0
    plt.figure(figsize=(15,6))

    with torch.no_grad():
        for images, targets in dataloader:
            images, targets = images.to(device), targets.to(device)
            outputs = model(images)
            preds = outputs.argmax(dim=1)

            for i in range(images.size(0)):
                if images_shown >= num_images:
                    break
                img = images[i].cpu().permute(1,2,0).numpy()
                # обратная нормализация
                img = np.clip(img * np.array([0.229,0.224,0.225]) + np.array([0.485,0.456,0.406]), 0, 1)
                true_label = CLASS_NAMES[targets[i].item()]
                pred_label = CLASS_NAMES[preds[i].item()]

                plt.subplot(2, num_images//2, images_shown+1)
                plt.imshow(img)
                plt.axis("off")
                plt.title(f"T: {true_label}\nP: {pred_label}", fontsize=10)
                images_shown += 1
            if images_shown >= num_images:
                break

    plt.tight_layout()
    plt.show()



def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--epochs', type=int, default=15)
    parser.add_argument('--batch-size', type=int, default=32 )
    parser.add_argument('--lr', type=float, default=0.01)
    parser.add_argument('--momentum', type=float, default=0.9)
    parser.add_argument('--workers', type=int, default=2)
    parser.add_argument('--save-model', type=str, default='stl10_densenet121.pth')
    args = parser.parse_args()
```

```
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print('Device:', device)

    train_loader, test_loader = get_dataloaders(batch_size=args.batch_size, num_workers=args.workers)

    model = DenseNet121Modified(num_classes=10).to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum,
weight_decay=1e-4)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)

    train_losses = []
    val_losses = []
    best_acc = 0.0

    for epoch in range(1, args.epochs+1):
        print(f'Epoch {epoch}/{args.epochs}')
        train_loss = train_one_epoch(model, device, train_loader, criterion, optimizer)
        val_loss, val_acc = evaluate(model, device, test_loader, criterion)

        train_losses.append(train_loss)
        val_losses.append(val_loss)

        print(f'  Train loss: {train_loss:.4f}')
        print(f'  Val loss:   {val_loss:.4f}, Val acc: {val_acc*100:.2f}%')

        if val_acc > best_acc:
            best_acc = val_acc
            torch.save(model.state_dict(), args.save_model)
            print(f'  Saved best model (acc={best_acc*100:.2f}%) to {args.save_model}')

        scheduler.step()

    plot_losses(train_losses, val_losses, out_path='loss_curve_densenet.png')

    show_predictions(model, device, test_loader, num_images=8)


if __name__ == '__main__':
    main()
```

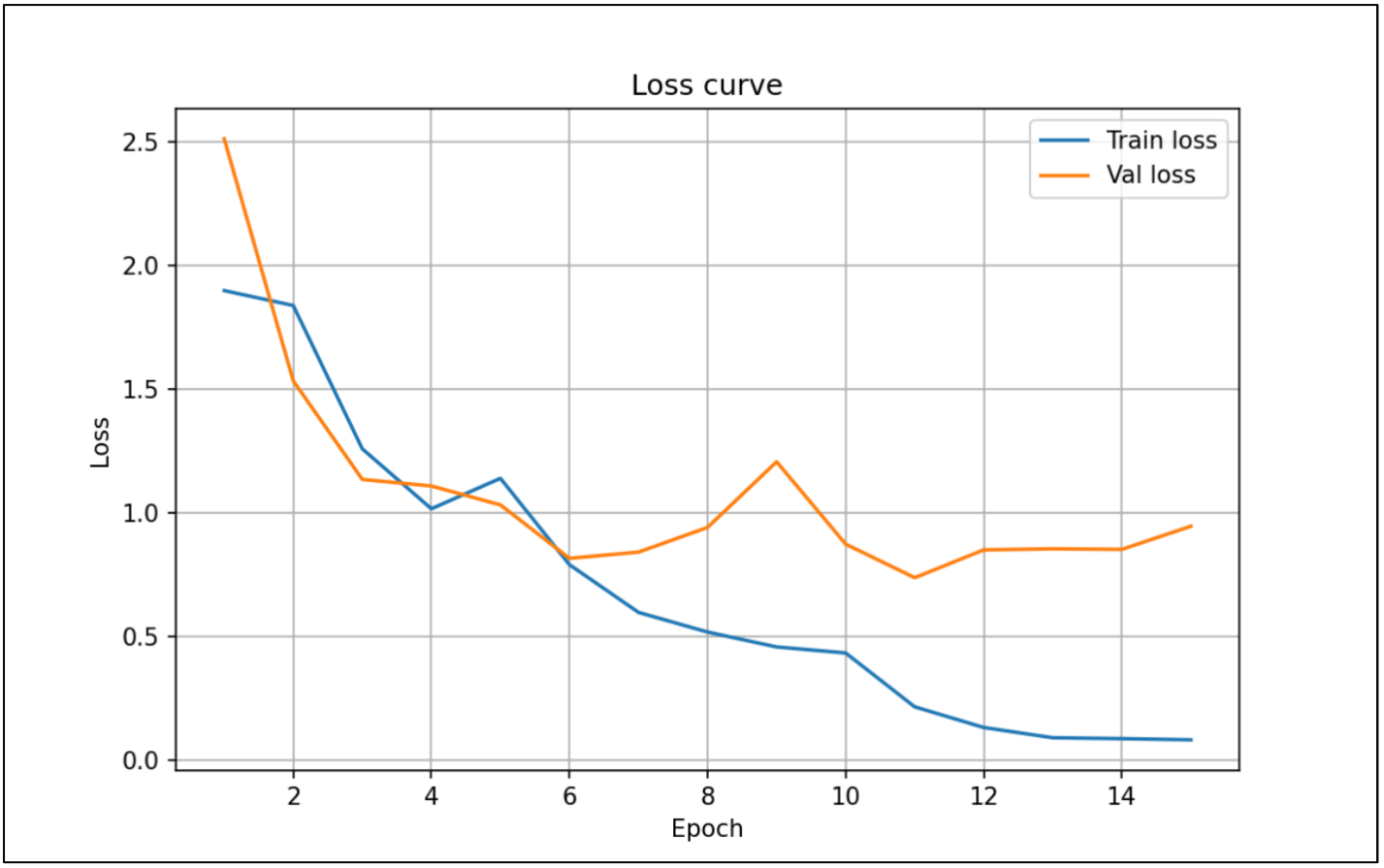
Результаты работы:

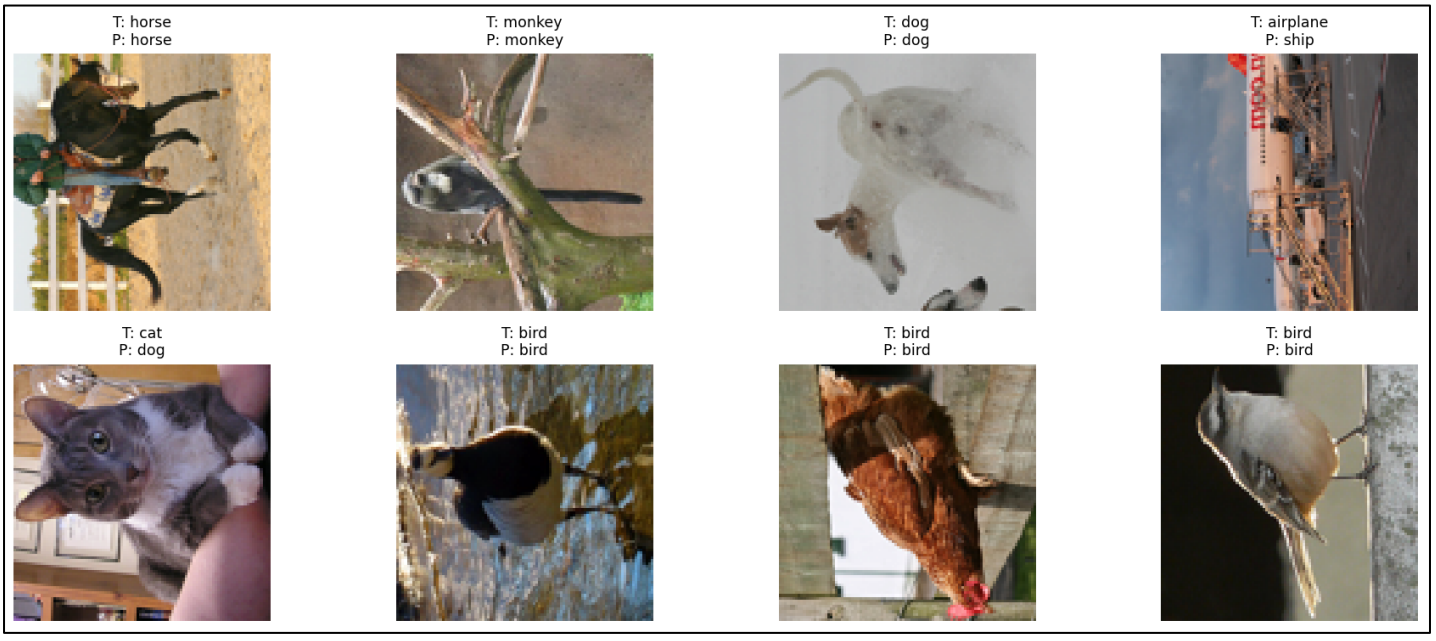
```
C:\Users\Asus\AppData\Local\Programs\Python\Python39\python.exe
"C:\Users\Asus\PycharmProjects\ОИИС\Лаба 2.py"
Device: cpu
Downloading: "https://download.pytorch.org/models/densenet121-a639ec97.pth" to
C:\Users\Asus/.cache\torch\hub\checkpoints\densenet121-a639ec97.pth
100%|██████████| 30.8M/30.8M [00:04<00:00, 6.64MB/s]
```

```
        Epoch 1/15
          Train loss: 1.8993
          Val loss:   2.5135, Val acc: 50.68%
        Train batches:   0%|          | 0/157 [00:00<?, ?it/s]  Saved best model (acc=50.68%) to
stl10_densenet121.pth
        Epoch 2/15
        Train batches:   0%|          | 0/157 [00:00<?, ?it/s]  Train loss: 1.8390
          Val loss:   1.5342, Val acc: 47.71%
        Epoch 3/15
          Train loss: 1.2594
          Val loss:   1.1359, Val acc: 64.40%
        Train batches:   0%|          | 0/157 [00:00<?, ?it/s]  Saved best model (acc=64.40%) to
stl10_densenet121.pth
        Epoch 4/15
        Train batches:   0%|          | 0/157 [00:00<?, ?it/s]  Train loss: 1.0172
          Val loss:   1.1092, Val acc: 63.52%
        Epoch 5/15
          Train loss: 1.1402
          Val loss:   1.0329, Val acc: 66.51%
        Train batches:   0%|          | 0/157 [00:00<?, ?it/s]  Saved best model (acc=66.51%) to
stl10_densenet121.pth
        Epoch 6/15
          Train loss: 0.7911
          Val loss:   0.8167, Val acc: 72.08%
          Saved best model (acc=72.08%) to stl10_densenet121.pth
        Epoch 7/15
        Train batches:   0%|          | 0/157 [00:00<?, ?it/s]  Train loss: 0.5981
          Val loss:   0.8419, Val acc: 70.86%
        Epoch 8/15
          Train loss: 0.5187
          Val loss:   0.9418, Val acc: 72.28%
          Saved best model (acc=72.28%) to stl10_densenet121.pth
        Epoch 9/15
        Train batches:   0%|          | 0/157 [00:00<?, ?it/s]  Train loss: 0.4583
          Val loss:   1.2073, Val acc: 68.20%
        Epoch 10/15
          Train loss: 0.4340
          Val loss:   0.8741, Val acc: 75.01%
          Saved best model (acc=75.01%) to stl10_densenet121.pth
        Epoch 11/15
          Train loss: 0.2162
          Val loss:   0.7385, Val acc: 78.99%
          Saved best model (acc=78.99%) to stl10_densenet121.pth
        Epoch 12/15
        Train batches:   0%|          | 0/157 [00:00<?, ?it/s]  Train loss: 0.1326
          Val loss:   0.8509, Val acc: 78.46%
        Epoch 13/15
          Train loss: 0.0913
          Val loss:   0.8552, Val acc: 79.71%
          Saved best model (acc=79.71%) to stl10_densenet121.pth
        Epoch 14/15
        Train batches:   0%|          | 0/157 [00:00<?, ?it/s]  Train loss: 0.0875
          Val loss:   0.8530, Val acc: 79.10%
        Epoch 15/15
          Train loss: 0.0828
```

Val loss: 0.9466, Val acc: 78.38%
Saved loss plot to loss_curve_densenet.png

Process finished with exit code 0





Сравнение с SOTA:

Ссылка на SOTA: https://hyper.ai/en/sota/tasks/image-classification/benchmark/image-classification-on-stl-10

Модель	**Архитектура**	**Тип обучения**	**Оптимизатор**	**Точность на тесте (%)**	**Отличие от SOTA**	**Возможные причины различий**
Моя модель	DenseNet-121	Дообучение (fine-tuning)	SGD (lr=0.01, momentum=0.9)	**79.1**	−11.0	Использование меньшего числа эпох, простая аугментация, отсутствие глубокого тюнинга гиперпараметров, ограниченный размер STL-10, обучение "в лоб" без регуляризаций и специальных приёмов.
DLME (SOTA)	ResNet-50	Обучение с продвинутым методом **Deep Local-flatness Manifold Embedding**	Не указано в статье	**90.1**	—	Использует продвинутые методы оптимизации представлений, специальные функции потерь и архитектурные улучшения, направленные на повышение устойчивости и обобщающей способности.

Вывод: сравнение с современными state-of-the-art решениями (например, DLME**,** ResNet-50) показывает, что точность моей модели ниже примерно на 11%. Это объясняется использованием более простой схемы дообучения без сложных методов оптимизации признаков, меньшим количеством эпох и ограниченными вычислительными ресурсами. Тем не менее, полученный результат можно считать удовлетворительным, так как предобученная DenseNet продемонстрировала хорошее обобщение и стабильное снижение функции потерь. Работа позволила на практике сравнить возможности кастомных архитектур и предобученных сверточных сетей, а также оценить влияние архитектуры и методов обучения на итоговую точность.