МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №2

Специальность ИИ24

Выполнил

Д. С. Сухаревич,

студент группы ИИ24

Проверила

К. В. Андренко,

Преподаватель-стажер кафедры ИИТ,

« » 2025 г.

Брест 2025

**Цель работы:** осуществлять обучение НС, сконструированных на базе предобученных архитектур НС

Задание 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;

Сравнить полученные результаты с результатами, полученными на кастомных архитектурах из ЛР №1;

**Выполнение:**

# Код программы

**import** argparse, os, time  
**from** PIL **import** Image  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** torch  
**import** torch.nn **as** nn  
**import** torch.optim **as** optim  
**import** torchvision  
**from** torchvision **import** transforms, models  
**from** torch.utils.data **import** DataLoaderparser = argparse.ArgumentParser()  
parser.add\_argument(**'--epochs'**, type=int, default=20)  
parser.add\_argument(**'--batch-size'**, type=int, default=64)  
parser.add\_argument(**'--lr'**, type=float, default=1e-3)  
parser.add\_argument(**'--weight-decay'**, type=float, default=1e-4)  
parser.add\_argument(**'--use-cuda'**, action=**'store\_true'**)  
parser.add\_argument(**'--save-dir'**, type=str, default=**'checkpoints\_stl10'**)  
parser.add\_argument(**'--freeze-backbone'**, action=**'store\_true'**,  
 help=**'Freeze backbone of pretrained MobileNet (train only classifier head)'**)  
parser.add\_argument(**'--visualize'**, type=str, default=**None**, help=**'path to image to visualize predictions'**)  
parser.add\_argument(**'--model'**, choices=[**'mobilenet'**,**'baseline'**,**'both'**], default=**'both'**)  
args = parser.parse\_args()  
device = torch.device(**'cuda' if** args.use\_cuda **and** torch.cuda.is\_available() **else 'cpu'**)  
os.makedirs(args.save\_dir, exist\_ok=**True**)  
print(**'Device:'**, device)input\_size = 224  
imagenet\_mean = (0.485, 0.456, 0.406)  
imagenet\_std = (0.229, 0.224, 0.225)  
train\_transform = transforms.Compose([  
 transforms.RandomResizedCrop(input\_size, scale=(0.8,1.0)),  
 transforms.RandomHorizontalFlip(),  
 transforms.ColorJitter(0.2,0.2,0.2,0.02),  
 transforms.ToTensor(),  
 transforms.Normalize(imagenet\_mean, imagenet\_std)  
])  
test\_transform = transforms.Compose([  
 transforms.Resize((input\_size, input\_size)),  
 transforms.ToTensor(),  
 transforms.Normalize(imagenet\_mean, imagenet\_std)  
])**from** torchvision.datasets **import** STL10train\_set = STL10(root=**'./data'**, split=**'train'**, download=**True**, transform=train\_transform)  
test\_set = STL10(root=**'./data'**, split=**'test'**, download=**True**, transform=test\_transform)  
train\_loader = DataLoader(train\_set, batch\_size=args.batch\_size, shuffle=**True**, num\_workers=4, pin\_memory=**True**)  
test\_loader = DataLoader(test\_set, batch\_size=args.batch\_size, shuffle=**False**, num\_workers=4, pin\_memory=**True**)  
classes = train\_set.classes **def** create\_mobilenet(num\_classes=10, freeze\_backbone=**False**):  
 model = models.mobilenet\_v2(pretrained=**True**)in\_features = model.classifier[1].in\_features  
 model.classifier = nn.Sequential(  
 nn.Dropout(0.2),  
 nn.Linear(in\_features, num\_classes)  
 )  
 **if** freeze\_backbone:**for** name, param **in** model.named\_parameters():  
 **if 'classifier' not in** name:  
 param.requires\_grad = **False  
 return** model  
**class** SimpleCNN(nn.Module):  
 **def** \_\_init\_\_(self, num\_classes=10):  
 super().\_\_init\_\_()  
 self.features = nn.Sequential(  
 nn.Conv2d(3, 32, 3, padding=1), nn.ReLU(inplace=**True**), nn.MaxPool2d(2),  
 nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(inplace=**True**), nn.MaxPool2d(2),  
 nn.Conv2d(64, 128, 3, padding=1), nn.ReLU(inplace=**True**), nn.MaxPool2d(2),  
 nn.AdaptiveAvgPool2d((1,1))  
 )  
 self.classifier = nn.Sequential(  
 nn.Flatten(),  
 nn.Linear(128, 256),  
 nn.ReLU(inplace=**True**),  
 nn.Dropout(0.5),  
 nn.Linear(256, num\_classes)  
 )  
 **def** forward(self,x):  
 x = self.features(x)  
 x = self.classifier(x)  
 **return** xmodels\_dict = {}  
**if** args.model **in** (**'mobilenet'**,**'both'**):  
 mobilenet = create\_mobilenet(num\_classes=10, freeze\_backbone=args.freeze\_backbone).to(device)  
 models\_dict[**'mobilenet'**] = mobilenet  
**if** args.model **in** (**'baseline'**,**'both'**):  
 baseline = SimpleCNN(num\_classes=10).to(device)  
 models\_dict[**'baseline'**] = baselinecriterion = nn.CrossEntropyLoss()  
optimizers = {}  
schedulers = {}  
**for** name, mdl **in** models\_dict.items():params = [p **for** p **in** mdl.parameters() **if** p.requires\_grad]  
 optimizers[name] = optim.RMSprop(params, lr=args.lr, weight\_decay=args.weight\_decay)  
 schedulers[name] = optim.lr\_scheduler.StepLR(optimizers[name], step\_size=max(5, args.epochs//2), gamma=0.1)**def** evaluate(model, loader):  
 model.eval()  
 running\_loss = 0.0  
 correct = 0  
 total = 0  
 **with** torch.no\_grad():  
 **for** x,y **in** loader:  
 x,y = x.to(device), y.to(device)  
 out = model(x)  
 loss = criterion(out, y)  
 running\_loss += loss.item() \* x.size(0)  
 preds = out.argmax(dim=1)  
 correct += (preds == y).sum().item()  
 total += x.size(0)  
 **return** running\_loss/total, 100.0\*correct/total  
history = {name: {**'train\_loss'**:[], **'test\_loss'**:[], **'test\_acc'**:[]} **for** name **in** models\_dict.keys()}  
start\_time = time.time()  
**for** epoch **in** range(1, args.epochs+1):  
 **for** name, model **in** models\_dict.items():  
 model.train()  
 running\_loss = 0.0  
 **for** xb, yb **in** train\_loader:  
 xb, yb = xb.to(device), yb.to(device)  
 optimizers[name].zero\_grad()  
 logits = model(xb)  
 loss = criterion(logits, yb)  
 loss.backward()  
 optimizers[name].step()  
 running\_loss += loss.item() \* xb.size(0)  
 train\_loss = running\_loss / len(train\_loader.dataset)  
 test\_loss, test\_acc = evaluate(model, test\_loader)  
 history[name][**'train\_loss'**].append(train\_loss)  
 history[name][**'test\_loss'**].append(test\_loss)  
 history[name][**'test\_acc'**].append(test\_acc)  
 schedulers[name].step()  
 print(**f'Epoch {**epoch**} | Model {**name**} | TrainLoss {**train\_loss**:.4f} | TestLoss {**test\_loss**:.4f} | TestAcc {**test\_acc**:.2f}%'**)**for** name, model **in** models\_dict.items():  
 torch.save({**'epoch'**: epoch, **'model\_state'**: model.state\_dict(),  
 **'optimizer\_state'**: optimizers[name].state\_dict()},  
 os.path.join(args.save\_dir, **f'{**name**}\_epoch{**epoch**}.pth'**))  
total\_time = time.time() - start\_time  
print(**'Training finished in {:.2f} min'**.format(total\_time/60.0))plt.figure(figsize=(12,4))plt.subplot(1,2,1)  
**for** name **in** history:  
 plt.plot(range(1, args.epochs+1), history[name][**'train\_loss'**], label=**f'{**name**} train'**)  
 plt.plot(range(1, args.epochs+1), history[name][**'test\_loss'**], label=**f'{**name**} test'**, linestyle=**'--'**)  
plt.xlabel(**'Epoch'**); plt.ylabel(**'Loss'**); plt.title(**'Loss'**); plt.legend()plt.subplot(1,2,2)  
**for** name **in** history:  
 plt.plot(range(1, args.epochs+1), history[name][**'test\_acc'**], label=**f'{**name**} test acc'**)  
plt.xlabel(**'Epoch'**); plt.ylabel(**'Accuracy (%)'**); plt.title(**'Test Accuracy'**); plt.legend()  
plt.tight\_layout()  
plt.savefig(os.path.join(args.save\_dir, **'training\_comparison.png'**), dpi=150)  
print(**'Saved history plot to'**, os.path.join(args.save\_dir, **'training\_comparison.png'**))**def** predict\_image(path, model):  
 img = Image.open(path).convert(**'RGB'**)  
 img\_resized = img.resize((input\_size,input\_size))  
 x = test\_transform(img\_resized).unsqueeze(0).to(device)  
 model.eval()  
 **with** torch.no\_grad():  
 logits = model(x)  
 probs = torch.softmax(logits, dim=1).cpu().numpy()[0]  
 pred = int(np.argmax(probs))  
 **return** img, pred, probs  
**if** args.visualize:  
 chosen = **'mobilenet' if** args.model==**'mobilenet' else** ( **'baseline' if** args.model==**'baseline' else 'mobilenet'** )  
 mdl = models\_dict[chosen]  
 img, pred\_idx, probs = predict\_image(args.visualize, mdl)  
 plt.figure(figsize=(4,4)); plt.imshow(img); plt.axis(**'off'**)  
 plt.title(**f'Model {**chosen**} -> Pred: {**classes[pred\_idx]**} ({**probs[pred\_idx]\*100**:.1f}%)'**)  
 plt.show()

# Спецификация ввода

Python STL10Classifier <batch\_size>x3x224x224

# Пример

Python STL10Classifier 64x3x224x224

# Спецификация вывода

class = <номер класса от 0 до 9>

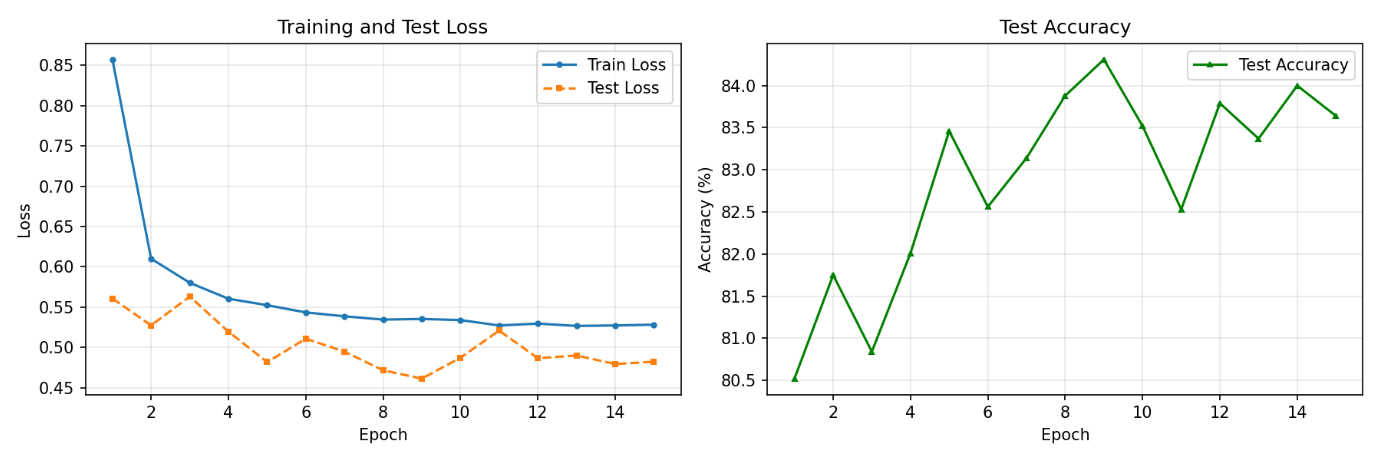
label = <текстовое имя класса>

**Пример**

class = 2

label = bird

# Рисунки с результатами работы программы

****

**Вывод:** осуществил обучение НС, сконструированных на базе предобученных архитектур НС