Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине «Обработка изображений в ИС»

Тема: «Конструирование моделей на базе предобученных нейронных сетей»

Выполнил:

Студент 4 курса

Группы ИИ-21

Шпак И.С.

Проверил:

Крощенко А.А.

Брест 2024

16	CIFAR-100	RMSprop	SqueezeNet 1.1

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

Код программы:

```
import torch
import torchvision # type ignore
from torchvision.transforms import v2
import torch.nn as nn
import numpy as np
import seaborn as sns
from tqdm import tqdm
import torch.nn.functional as F
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix


batch_size_train = 64
batch_size_test = 100


preprocess = v2.Compose([
  v2.Resize(256),
  v2.CenterCrop(224),
  v2.ToTensor(),
  v2.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
train_loader = torch.utils.data.DataLoader(
```

```
    torchvision.datasets.CIFAR100(root='./data', train=True, download=True,
                    transform=preprocess), batch_size=batch_size_train, shuffle=True
  )
test_loader = torch.utils.data.DataLoader(
    torchvision.datasets.CIFAR100(root='./data', train=False, download=True,
                    transform=preprocess), batch_size=batch_size_test, shuffle=False
  )

model = torch.hub.load('pytorch/vision v0.10.0', 'squeezenet1_1', weights="IMAGENET1K_V1")

#model.classifier._modules["1"] = nn.Conv2d(512, 100, kernel_size=(1, 1))
model.classifier[1].out_channels = 100
model.classifier[0] = nn.BatchNorm2d(512)
model

def train(device, model, train_loader, learning_rate=0.001, epochs=5, model_save_path='best_model.pth')
  loss_fn = nn.CrossEntropyLoss().to(device)
  for param in model.classifier.parameters()
    param.requires_grad = True
  trainable_params = [p for p in model.parameters() if p.requires_grad==True]
  optimizer = torch.optim.RMSprop(trainable_params, lr=learning_rate)
  history = []
  best_loss = float('inf')
```

```
for epoch in tqdm(range(epochs))
    epoch_loss = 0.0
    for x, y in train_loader
        optimizer.zero_grad()
        x, y = x.to(device), y.to(device)
        pred = model(x)

        loss = loss_fn(pred, y)
        epoch_loss += loss.item()
        loss.backward()
        optimizer.step()

    average_loss = epoch_loss / len(train_loader)
    history.append(average_loss)

    if average_loss < best_loss
        best_loss = average_loss
        torch.save(model.state_dict(), model_save_path)
        print(f'Model saved with loss {best_loss} at epoch {epoch + 1}')

    print(f'Epoch {epoch + 1}, Loss  {average_loss}')

plt.plot(range(0, epochs), history)
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
    plt.title('Training Loss per Epoch')
    plt.show()

def test(model, device, test_loader)
    model.eval()
    correct = 0
    total = 0
    all_labels = []
    all_predictions = []
    num_classes = 100
    with torch.no_grad()
        for images, labels in test_loader
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)

            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    accuracy = correct / total
    print(f"Accuracy on the test set {accuracy .2%}")
```

```
    cm = confusion_matrix(all_labels, all_predictions)
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[ , np.newaxis]

    plt.figure(figsize=(20, 18))
    sns.heatmap(cm_normalized, annot=False, fmt='.2f', cmap='Blues', cbar=True)

    plt.xlabel('Predicted', fontsize=14)
    plt.ylabel('True', fontsize=14)
    plt.title('Confusion Matrix (Normalized)', fontsize=16)

    plt.xticks(np.arange(num_classes) + 0.5, labels=np.arange(num_classes), rotation=90, fontsize=10)
    plt.yticks(np.arange(num_classes) + 0.5, labels=np.arange(num_classes), rotation=0, fontsize=10)

    plt.tight_layout()
    plt.show()

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
for param in model.parameters()
    param.requires_grad = False
model = model.to(device)

model.load_state_dict(torch.load('best_model.pth'))

train(device, model, train_loader, learning_rate=10e-4, epochs=10)
```

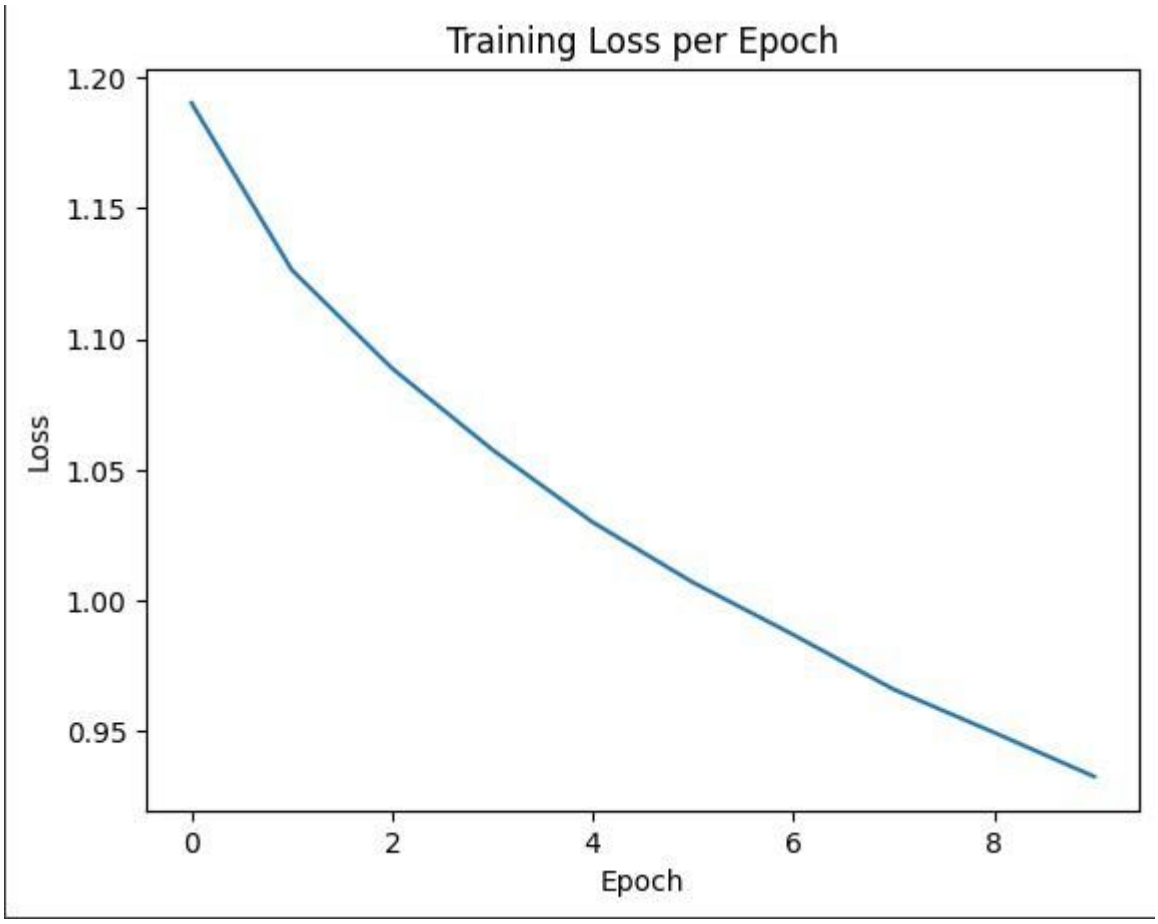
```
test(model, device, test_loader)


def validate(model, device, test_loader)
    model.eval()
    with torch.no_grad()
        dataset = test_loader.dataset
        random_index = np.random.randint(0, len(dataset))
        single_example = dataset[random_index]
        images, labels = single_example[0], single_example[1]
        images = images.unsqueeze(0).to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        print(f"Предсказано  {predicted.item()}, Реально  {labels}")



dataset = test_loader.dataset
random_index = int(np.random.random()*len(dataset))
single_example = dataset[random_index]
validate(model, device, test_loader)
```

График изменения ошибки при обучении



Точность на тестовой выборке

Точность на предобученной модели

```
Accuracy on the test set: 62.99%
```

Точность на непредобученной модели

```
Accuracy on the test set: 20.54%
```

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