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

Корнейчук А. И.

Проверил:

Крощенко А.А.

Брест 2024

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

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | CIFAR-100 | Adadelta | ResNet18 |

Код программы:

import torch

import torch.nn as nn

import torch.optim as optim

import matplotlib.pyplot as plt

import numpy as np

from torchvision import datasets, transforms, models

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

batch\_size = 64

transform = transforms.Compose([

transforms.Resize((224, 224)),

transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))

])

train\_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)

test\_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

train\_loader = torch.utils.data.DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)

test\_loader = torch.utils.data.DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False)

learning\_rate = 1.0

model = models.resnet18(pretrained=True)

model.fc = nn.Linear(model.fc.in\_features, 10)

model = model.to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adadelta(model.parameters(), lr=learning\_rate)

num\_epochs = 10

def train(model, train\_loader, criterion, optimizer, num\_epochs):

model.train()

train\_loss\_history = []

for epoch in range(num\_epochs):

running\_loss = 0.0

for images, labels in train\_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()

epoch\_loss = running\_loss / len(train\_loader)

train\_loss\_history.append(epoch\_loss)

print(f'Epoch [{epoch + 1}/{num\_epochs}], Loss: {epoch\_loss:.4f}')

return train\_loss\_history

train\_loss\_history = train(model, train\_loader, criterion, optimizer, num\_epochs)

plt.plot(train\_loss\_history, label='Training Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

from sklearn.metrics import confusion\_matrix

import seaborn as sns

def evaluate(model, test\_loader):

model.eval()

correct = 0

total = 0

all\_predictions = []

all\_labels = []

num\_classes = 10

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\_predictions.extend(predicted.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

accuracy = 100 \* correct / total

print(f'Test Accuracy: {accuracy:.2f}%')

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()

return accuracy

test\_accuracy = evaluate(model, test\_loader)

def visualize\_predictions(model, test\_loader, num\_images=5):

model.eval()

images\_shown = 0

class\_names = test\_dataset.classes

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)

for i in range(images.size(0)):

if images\_shown == num\_images:

return

img = images[i].cpu().numpy().transpose((1, 2, 0))

img = (img \* 0.5 + 0.5)

plt.imshow(img)

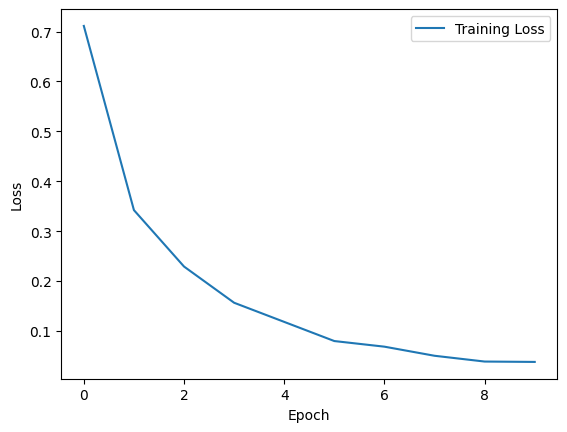
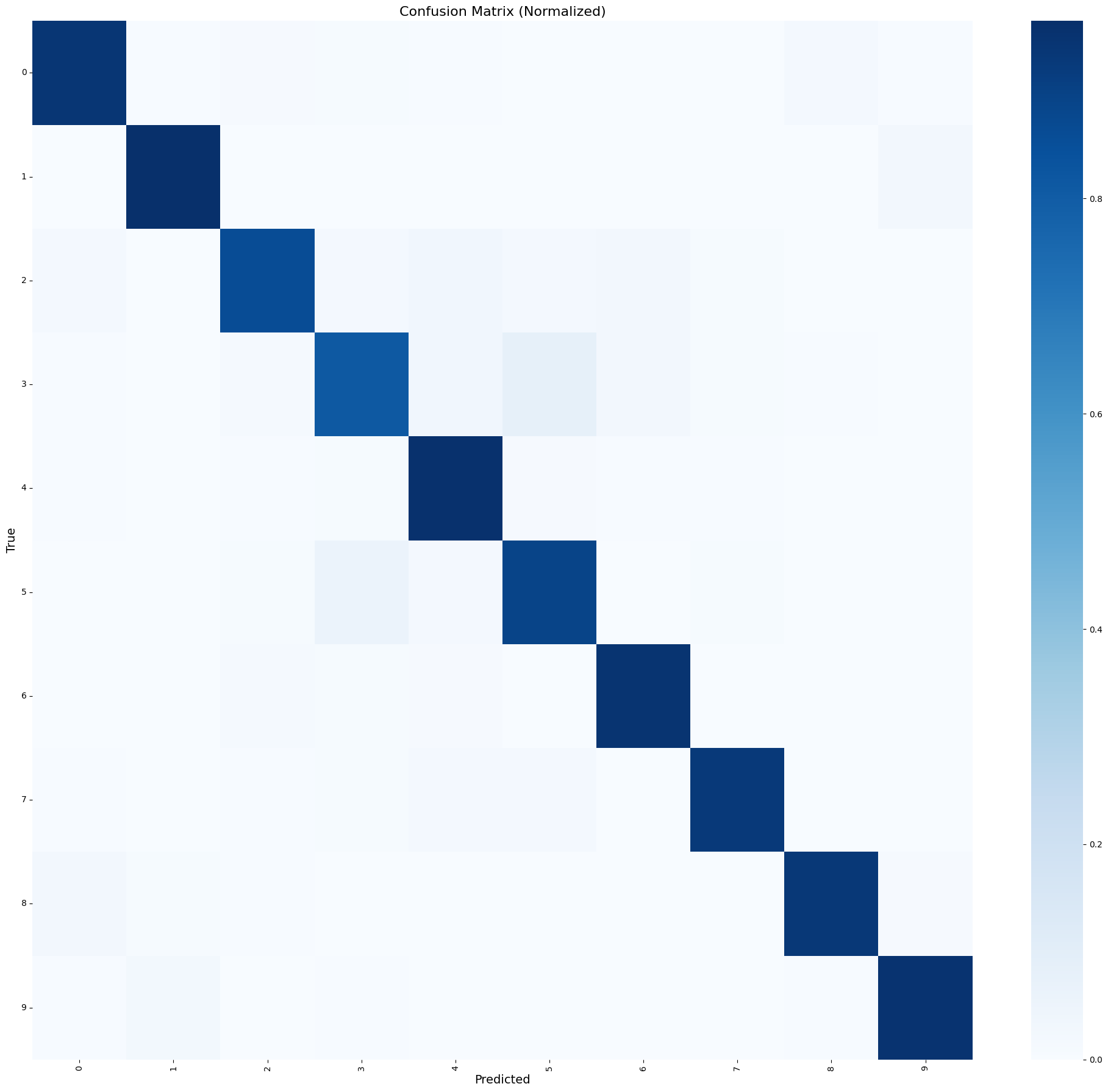
plt.title(f'Predicted: {class\_names[predicted[i]]}, Actual: {class\_names[labels[i]]}')

plt.axis('off')

plt.show()

images\_shown += 1

visualize\_predictions(model, test\_loader)





**Вывод**: научился осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.