ГУАП

КАФЕДРА № 14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		Н.Н. Эпаев
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

ОБУЧЕНИЕ ГЛУБОКОЙ НЕЙРОННОЙ СЕТИ ДЛЯ РАСПОЗНАВАНИЯ ИЗОБРАЖЕНИЙ

по курсу: НЕЙРОННЫЕ СЕТИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 1045		Бондарев К. А.
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

1. Цель работы

Исследовать влияния размера обучающей выборки и архитектуры нейронной сети при обучении ее распознаванию изображений.

2. Теоретический минимум

С помощью библиотеки *pytorch* была реализована простая сверточная нейронная сеть с помощью нескольких стандартных классов из подмодуля *torch.nn*.

Сеть *ConvNet* имеет четыре слоя, два слоя сверточных, после каждого из которых применяется операция подвыборки (*maxpooling*) в окне 2×2, а также два полносвязных. В первом слое используется 32 сверточных фильтра, размером 5×5, во втором 64 фильтра того же размера. После сверточных идет два полносвязных слоя (*nn.Linear*) со 120 и 10 нейронами, соответственно. Во время инициализации нейронной сети (функция *__init__*) происходит выделение памяти под веса нейронной сети и присвоение им случайных начальных значений. Функция *forward*, принимает на вход тензор *pytorch*, содержащий изображение, а возвращает вектор, каждый элемент которого соответствует 10 классам изображений. Эта функция отвечает за порядок операций над изображением. Сначала выполняется свертка с ядрами *conv1*, потом применяется активационная функция (*relu*), а затем операция подвыборки. То же самое повторяется для второго сверточного слоя, после которого карта признаков с помощью команды *view* вытягивается в одномерный вектор. После последнего полносвязного слоя (*fc2*) функция возвращает предсказание-вектор, где большие значения соответствуют большей вероятности (в используемой далее функции потерь этот вектор будет автоматически отнормирован в распределение вероятностей).

3. Ход работы

Были загружена и инициализирована база данных изображений для обучения нейронной сети CIFAR-10 с помощью встроенных команд *pytorch*.

База данных содержит «крошечные» цветные изображения 10 классов размером 32×32 пикселя. Общее количество изображений 60 000, из них 50 000 используется для обучения, а 10 000 для тестирования.

Было визуализировано 10 изображений (рисунок 1).

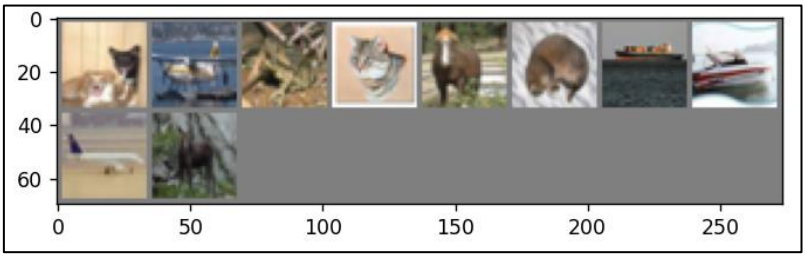

Рисунок 1 – Результат работы модели

3.1 Обучение нейронных сетей

Было выполнено обучение сети (рисунок 2).

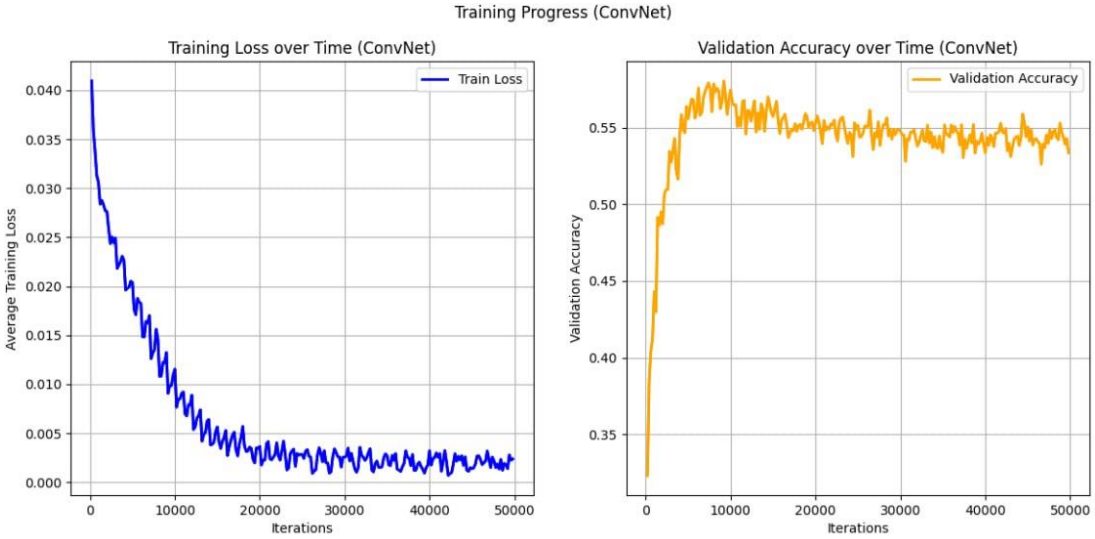

Рисунок 2 – Значение ошибки и точность распознавания валидационной выборки в процессе обучения сверточной нейронной сети

Как видно на рисунке 2, ошибка на каждой эпохе в среднем уменьшается, что свидетельствует об успешном процессе обучения. Однако на каждой итерации наблюдаются значительные выбросы, что объясняется разной сложностью изображений — некоторые из них легче для распознавания, а другие сложнее.

Также из рисунка можно заметить, что точность в начале обучения быстро увеличивается и достигает пика примерно к 8 тысячам итераций, после чего начинает медленно снижаться, что указывает на переобучение после 8-10 тысяч итераций. Для практического применения предпочтительно использовать веса нейронной сети, полученные примерно на 8 тысячах итераций. Согласно графику, сеть в среднем корректно распознает около 5 изображений из 10.

Чтобы оценить преимущества современных сверточных нейронных сетей по сравнению с другими архитектурами, создадим простую полносвязную сеть и проведем аналогичные эксперименты с ней.

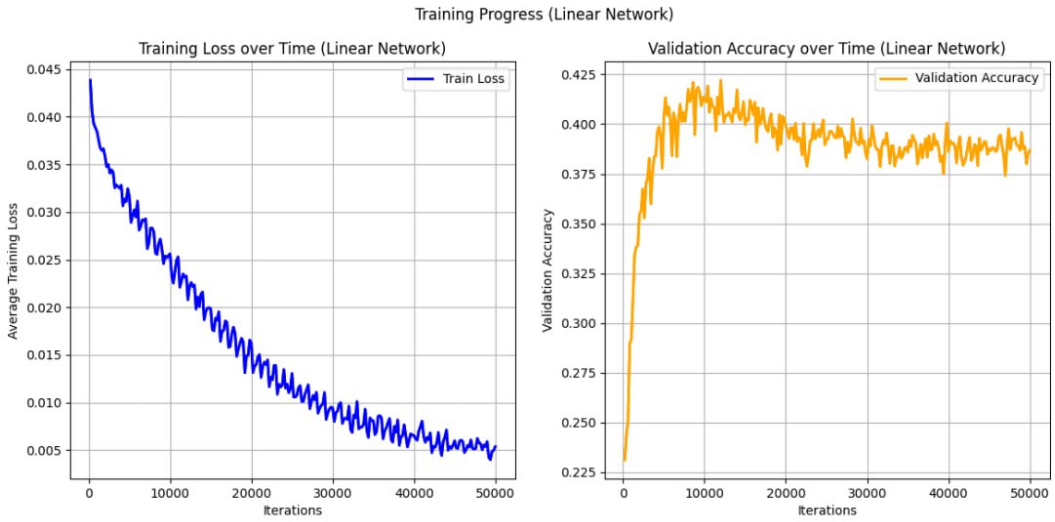

Рисунок 3 – Значение ошибки и точность распознавания валидационной выборки в процессе обучения полносвязной нейронной сети

На рисунке 3 видно, что на каждой эпохе ошибка в среднем уменьшается, что подтверждает успешное обучение. График показывает значительные выбросы на каждой итерации, но в сравнении с графиком сверточной сети он выглядит более плавным, с меньшими выбросами, хотя снижение ошибки происходит медленнее.

На рисунке также можно заметить, что точность в начале обучения быстро возрастает, достигая пика примерно к 11 тысячам итераций, после чего начинает постепенно снижаться, что свидетельствует о начале переобучения после 11 тысяч итераций. Из графика видно, что сеть в среднем распознает чуть больше 4 изображений из 10. В сравнении со сверточной нейросетью, период до переобучения больше, но точность распознавания ниже.

3.2 Изменение параметров

Увеличим количество обучающей выборки в 5 раз.

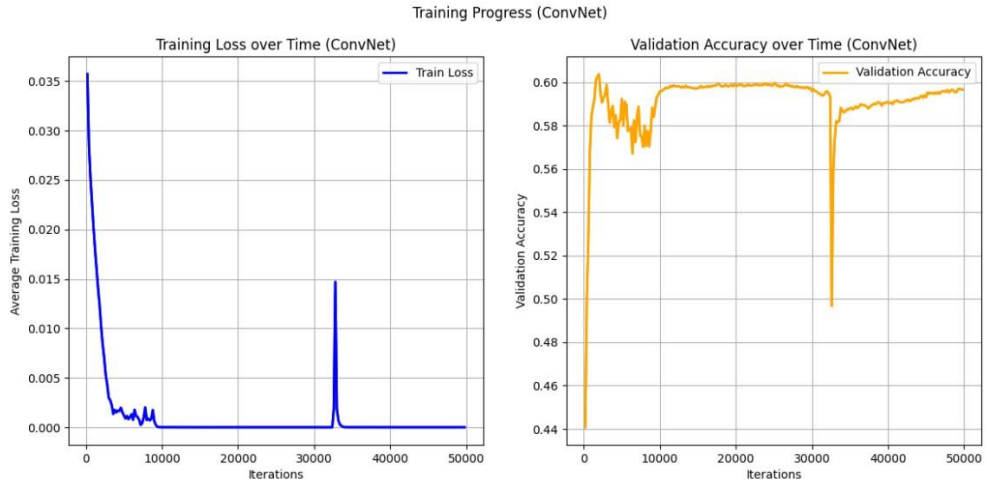

Рисунок 4 – Значение ошибки и точность распознавания в процессе обучения сверточной нейронной сети при увеличении обучающей выборки 5 раз

```
train_loss:
iteration 49000. accuracy: 0.5954
train_loss: 2.268313477671313e-09
iteration 49200. accuracy: 0.5967
train_loss: 2.038239729174052e-09
iteration 49400. accuracy: 0.5969
train_loss: 1.837968246576338e-09
iteration 49600. accuracy: 0.5968
train_loss: 1.6567702523673234e-09
iteration 49800. accuracy: 0.5966
train_loss: 1.501083010246873e-09
iteration 50000. accuracy: 0.5965
Start Training for Linear Network
train_loss: 0.0416922456741333
iteration 200. accuracy: 0.2766
train_loss: 0.03757072608470917
iteration 400. accuracy: 0.3266
```

```
train_loss: 2.268313477671313e-09
iteration 49200. accuracy: 0.5967
train_loss: 2.038239729174052e-09
iteration 49400. accuracy: 0.5969
train_loss: 1.837968246576338e-09
iteration 49600. accuracy: 0.5968
train_loss: 1.6567702523673234e-09
iteration 49800. accuracy: 0.5966
train_loss: 1.501083010246873e-09
iteration 50000. accuracy: 0.5965
Start Training for Linear Network
train_loss: 0.0416922456741333
iteration 200. accuracy: 0.2766
train_loss: 0.03757072608470917
iteration 400. accuracy: 0.3266
```

Рисунок 5-6 – Вывод в консоли

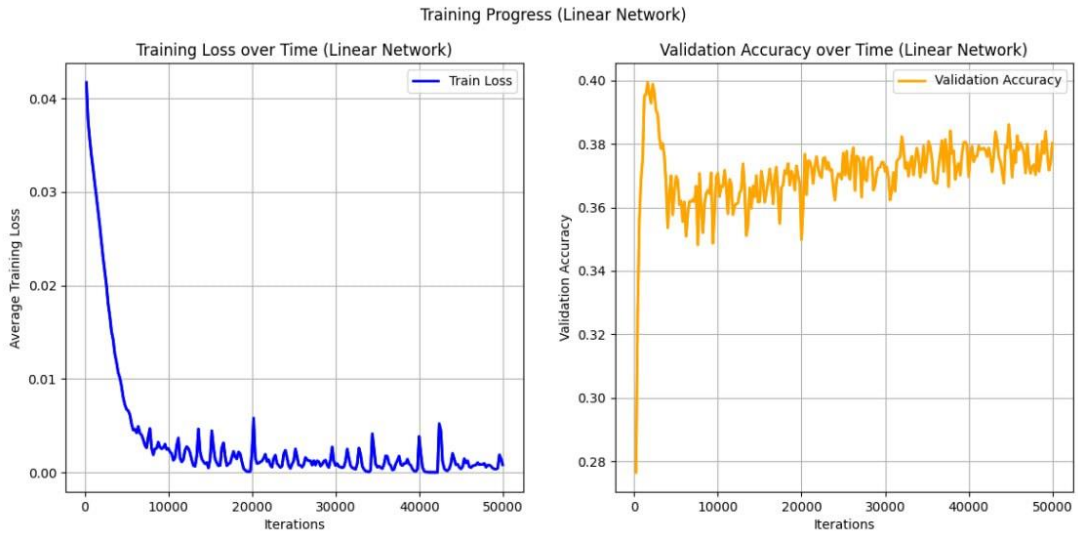

Рисунок 7 – Значение ошибки и точность распознавания в процессе обучения полносвязной нейронной сети при увеличении обучающей выборки 5 раз

Увеличим количество слоёв в каждой нейронной сети.

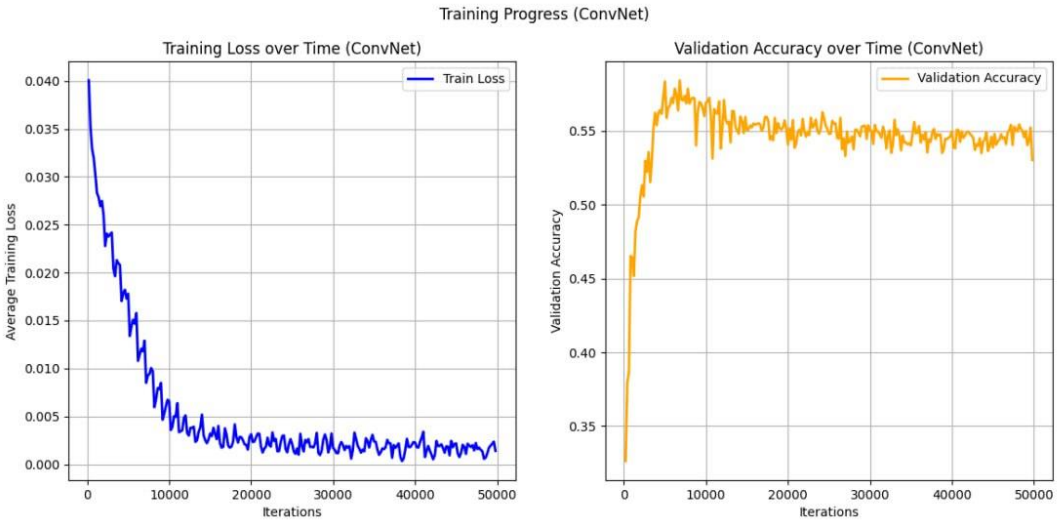

Рисунок 8 – Значение ошибки и точность распознавания в процессе обучения сверточной нейронной сети при добавлении ещё одного слоя

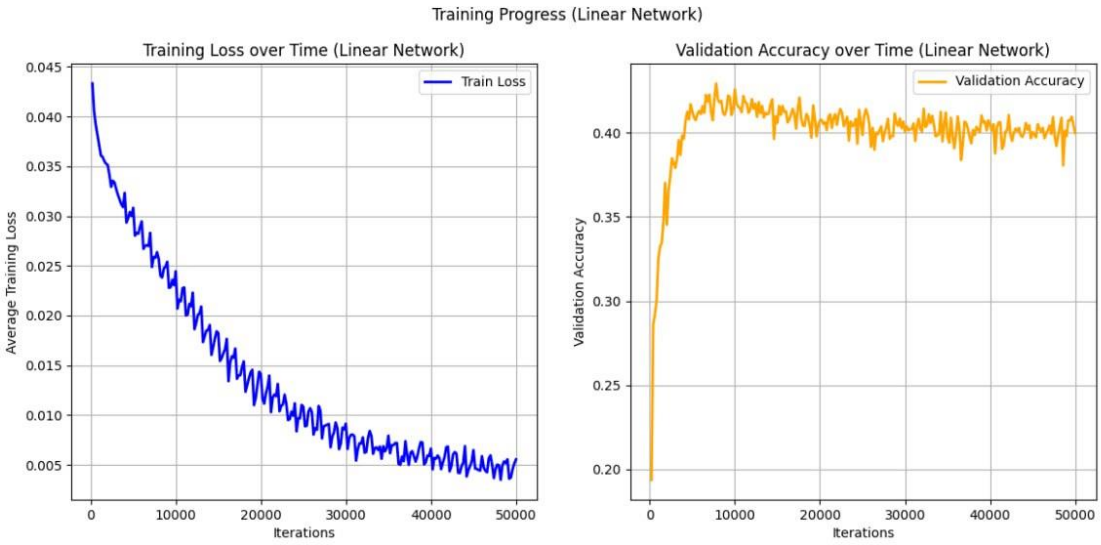

Рисунок 9 – Значение ошибки и точность распознавания в процессе обучения полносвязной нейронной сети при удалении одного слоя

4. Выводы

Таким образом, сверточная нейронная сеть начинает проявлять признаки переобучения. Сеть распознает в среднем немного меньше 6 изображений из 10, что свидетельствует об улучшении ее эффективности.

Для полносвязной нейронной сети при заданных итерациях переобучение не наблюдается. Она распознает в среднем чуть больше 4 изображений из 10, что указывает на отсутствие улучшений в точности.

Добавление дополнительных сверточных слоев значительно повышает качество распознавания изображений, поскольку сеть лучше выделяет ключевые признаки. Точность распознавания сначала растет, а затем стабилизируется на определенном уровне, при котором дальнейшее увеличение глубины уже не приводит к заметным улучшениям.

Приложение 1.

Листинг программы

```
import torchvision
import torch
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
from tensorboardX import SummaryWriter
import os
import matplotlib.pyplot as plt
import numpy as np
import torch.nn.functional as F
import multiprocessing
from queue import Empty

# Определение архитектур нейронных сетей
class ConvNet(nn.Module):
  def __init__(self):
    super(ConvNet, self).__init__()
    self.conv1 = nn.Conv2d(3, 32, 5)       # Первый сверточный слой
    self.pool = nn.MaxPool2d(2, 2)         # Слой подвыборки (пулинг)
    self.conv2 = nn.Conv2d(32, 64, 5)      # Второй сверточный слой
    self.fc1 = nn.Linear(64 * 5 * 5, 200)  # Первый полносвязный слой (увеличено до 200
нейронов)
    self.fc2 = nn.Linear(200, 10)          # Второй полносвязный слой (выходной слой)

  def forward(self, data):
    x = data
    x = self.pool(F.relu(self.conv1(x)))   # Применение первого сверточного слоя и пулинга
    x = self.pool(F.relu(self.conv2(x)))   # Применение второго сверточного слоя и пулинга
    x = x.view(-1, 64 * 5 * 5)             # Преобразование тензора для полносвязного слоя
    x = F.relu(self.fc1(x))                # Применение первого полносвязного слоя
    x = self.fc2(x)                        # Применение второго полносвязного слоя
    return x

class LinearNetwork(nn.Module):
  def __init__(self):
    super(LinearNetwork, self).__init__()
    self.fc1 = nn.Linear(32 * 32 * 3, 480)  # Первый полносвязный слой (увеличено до 480
нейронов)
    self.fc2 = nn.Linear(480, 180)          # Второй полносвязный слой (увеличено до 180
нейронов)
    self.fc3 = nn.Linear(180, 126)          # Третий полносвязный слой (увеличено до 126
нейронов)
    self.fc4 = nn.Linear(126, 10)           # Четвёртый полносвязный слой (выходной слой)
```

```
    def forward(self, data):
        x = data
        x = x.view(-1, 32 * 32 * 3)            # Преобразование тензора для полносвязных слоёв
        x = torch.sigmoid(self.fc1(x))         # Применение первого полносвязного слоя с функцией активации
        x = torch.sigmoid(self.fc2(x))         # Применение второго полносвязного слоя с функцией активации
        x = torch.sigmoid(self.fc3(x))         # Применение третьего полносвязного слоя с функцией активации
        x = self.fc4(x)                    # Применение четвёртого полносвязного слоя
        return x

# Функция для отображения изображений с помощью matplotlib
def imshow(img):
    img = img / 2 + 0.5  # Денормализация
    npimg = img.numpy()
    plt.figure(figsize=(8, 8))
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.title("Training Images")
    plt.axis('off')
    plt.show(block=False)  # Не блокировать основной поток

# Функция для вычисления точности на валидационном наборе
def get_validation_accuracy(net, device, testloader):
    num = 0
    correct = 0
    net.eval()  # Перевести модель в режим оценки
    with torch.no_grad():
        for data in testloader:
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)  # Переместить данные на устройство
            outputs = net(inputs)
            _, predicted = torch.max(outputs, 1)
            correct += torch.sum(predicted == labels).item()
            num += labels.size(0)  # Количество должно увеличиваться на размер батча
    net.train()  # Вернуть модель в режим обучения
    return float(correct) / num

# Функция для построения графиков в отдельном процессе
def plot_process(queue, terminate_event, model_name):
    plt.ion()  # Включить интерактивный режим
    fig, (ax_loss, ax_acc) = plt.subplots(1, 2, figsize=(14, 6))
    fig.suptitle(f'Training Progress ({model_name})')

    train_losses = []
```

```
    validation_acc = []
    iterations = []
    while not terminate_event.is_set():
        try:
            # Ожидание данных с таймаутом
            data = queue.get(timeout=0.1)
            if data == 'TERMINATE':
                break
            iter_num, avg_loss, acc = data
            iterations.append(iter_num)
            train_losses.append(avg_loss)
            validation_acc.append(acc)

            # Обновление графика Training Loss
            ax_loss.clear()
            ax_loss.plot(iterations, train_losses, linestyle='-', color='blue', linewidth=2, label='Train
Loss')
            ax_loss.set_title(f'Training Loss over Time ({model_name})')
            ax_loss.set_xlabel('Iterations')
            ax_loss.set_ylabel('Average Training Loss')
            ax_loss.grid(True)
            ax_loss.legend()

            # Обновление графика Validation Accuracy
            ax_acc.clear()
            ax_acc.plot(iterations, validation_acc, linestyle='-', color='orange', linewidth=2,
label='Validation Accuracy')
            ax_acc.set_title(f'Validation Accuracy over Time ({model_name})')
            ax_acc.set_xlabel('Iterations')
            ax_acc.set_ylabel('Validation Accuracy')
            ax_acc.grid(True)
            ax_acc.legend()

            plt.draw()
            plt.pause(0.001)  # Обработка событий GUI
        except Empty:
            continue
        except Exception as e:
            print(f"Ошибка в процессе построения графиков для {model_name}: {e}")
            break

    plt.ioff()
    plt.show()

def main():
    # Определение устройства
```

```
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Используемое устройство: {device}")

    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform)
    trainset = torch.utils.data.Subset(trainset, list(range(0, 10000)))
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=10, shuffle=True,
num_workers=0)

    testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform)
    testloader = torch.utils.data.DataLoader(testset, batch_size=50, shuffle=False,
num_workers=0)

    # Визуализация нескольких обучающих изображений с matplotlib
    try:
        dataiter = iter(trainloader)
        images, labels = next(dataiter)
        imshow(torchvision.utils.make_grid(images))
    except Exception as e:
        print(f"Ошибка при отображении изображений: {e}")

    # Создание списка сетей
    nets = []
    nets.append(ConvNet().to(device))  # Переместить модель на устройство
    nets.append(LinearNetwork().to(device))

    # Список процессов и очередей для графиков
    processes = []
    queues = []
    terminate_events = []

    for cnt, net in enumerate(nets):
        model_name = 'ConvNet' if cnt == 0 else 'Linear Network'
        queue = multiprocessing.Queue()
        terminate_event = multiprocessing.Event()
        p = multiprocessing.Process(target=plot_process, args=(queue, terminate_event,
model_name))
        p.start()
        processes.append(p)
        queues.append(queue)
```

```
        terminate_events.append(terminate_event)

    try:
        for cnt, net in enumerate(nets):
            model_name = 'ConvNet' if cnt == 0 else 'Linear Network'
            criterion = nn.CrossEntropyLoss()
            optimizer = optim.Adam(net.parameters())
            iterations = 50000  # Увеличено количество итераций

            log_folder = f'{model_name}_10000'
            log_dir = os.path.join('log', log_folder)
            writer = SummaryWriter(log_dir)

            print(f'Start Training for {model_name}')
            iteration = 0
            running_loss = 0.0
            validation_acc = []
            train_losses = []

            delimeter = 200  # Переменная для интервала логирования и валидации

            while True:
                if iteration >= iterations:
                    break


                for data in trainloader:
                    inputs, indexes = data
                    inputs, indexes = inputs.to(device), indexes.to(device)  # Переместить данные на
устройство

                    optimizer.zero_grad()
                    outputs = net(inputs)
                    loss = criterion(outputs, indexes)
                    loss.backward()
                    optimizer.step()

                    writer.add_scalar('train_loss', scalar_value=loss.item(), global_step=iteration)
                    running_loss += loss.item()
                    iteration += 1

                    if iteration % delimeter == 0:
                        avg_loss = running_loss / len(trainset)
                        train_losses.append(avg_loss)
                        print(f'train_loss: {avg_loss}')
                        writer.add_scalar('avg_train_loss', scalar_value=avg_loss, global_step=iteration)
```

```
                running_loss = 0.0

                acc = get_validation_accuracy(net, device, testloader)  # Передать модель и
устройство
                validation_acc.append(acc)
                print(f'iteration {iteration}. accuracy: {acc}')
                writer.add_scalar('val_accuracy', scalar_value=acc, global_step=iteration)

                # Отправка данных в очередь для построения графиков
                queues[cnt].put((iteration, avg_loss, acc))

            if iteration >= iterations:
                break

    except KeyboardInterrupt:
        print("\nПрограмма прервана пользователем (Ctrl+C). Выполняется очистка
ресурсов...")
    except Exception as e:
        print(f"\nПроизошла ошибка: {e}")
    finally:
        # Отправка сигнала завершения для каждого процесса построения графиков
        for q in queues:
            q.put('TERMINATE')

        # Установка события завершения
        for event in terminate_events:
            event.set()

        # Закрытие очередей
        for q in queues:
            q.close()

        # Ожидание завершения процессов построения графиков
        for p in processes:
            p.join()

        # Очистка кэша CUDA
        torch.cuda.empty_cache()

        print("Все ресурсы очищены. Программа завершена.")

if __name__ == "__main__":
    # Убедитесь, что код запускается в основном процессе
    multiprocessing.set_start_method('spawn')  # Необходимо для Windows
    main()
```