|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

Институт Информационных Технологий

Кафедра Вычислительной Техники (ВТ)

**ОТЧЁТ ПО ПРАКТИЧЕСКИМ РАБОТАМ**

по дисциплине

«Проектирование и обучение нейронных сетей»

|  |  |
| --- | --- |
| Выполнил студент группы:  ИКБО-15-22 | Оганнисян Г.А. |
| Принял старший преподаватель кафедры ВТ | Семенов Р.Э. |
| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_2025 г. |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2025 г. |

Москва 2025 г.

ТИТУЛЬНИК 2

ТИТУЛЬНИК 3

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc197254280)

[1 СРАВНИТЕЛЬНЫЙ АНАЛИЗ ЭФФЕКТИВНОСТИ СВЕРТОЧНЫХ СЕТЕЙ И ТРАНСФОРМЕРОВ В ЗАДАЧАХ КОМПЬЮТЕРНОГО ЗРЕНИЯ 6](#_Toc197254281)

[**1.1** **Введение и цель работы** 6](#_Toc197254282)

[**1.2** **Методика реализации** 6](#_Toc197254283)

[**1.2.1** **Реализация сверточной нейросети** 6](#_Toc197254284)

[**1.2.2** **Реализация трансформерной модели** 7](#_Toc197254285)

[**1.3** **Эксперементальная часть** 7](#_Toc197254286)

[**1.3.1** **Датасет** 7](#_Toc197254287)

[**1.3.2** **Параметры моделей** 8](#_Toc197254288)

[**1.3.3** **Результаты обучения** 8](#_Toc197254289)

[**1.4** **Сравнительный анализ** 11](#_Toc197254290)

[**1.5** **Примеры результатов предсказания** 11](#_Toc197254291)

[**1.6** **Вывод и заключение** 14](#_Toc197254292)

[ЗАКЛЮЧЕНИЕ 15](#_Toc197254293)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 16](#_Toc197254294)

[ПРИЛОЖЕНИЯ 17](#_Toc197254295)

**ВВЕДЕНИЕ**

В последние годы нейронные сети стали основой многих современных решений в области искусственного интеллекта, позволяя эффективно решать задачи обработки изображений, генерации контента, анализа графовых структур и многое другое. Развитие аппаратных ресурсов и появление новых архитектур нейронных сетей привели к появлению специализированных методов обучения, ориентированных на различные типы данных и прикладные задачи. В рамках данного отчёта рассматриваются четыре подхода к обучению нейронных сетей: метод «Трансформер», который использует механизмы внимания для глобального анализа входных данных; метод «Генеративная сеть (DCGAN)», применяемый для генерации новых образцов данных с помощью состязательной игры генератора и дискриминатора; метод «Графовые нейронные сети (GCN)», позволяющий эффективно работать с данными в виде графов; а также сравнительный анализ сверточных нейросетей и трансформеров в задаче классификации изображений.

Изучение различных алгоритмов обучения позволяет выбрать наиболее подходящий метод для конкретной задачи, учитывая особенности данных и требования к модели. Кроме того, понимание принципов работы различных методов обучения способствует разработке более эффективных и устойчивых нейронных сетей, способных справляться с разнообразными вызовами в реальных приложениях.

Таким образом, исследование и разработка методов обучения нейронных сетей продолжают оставаться важным направлением в области искусственного интеллекта, открывая новые возможности для создания интеллектуальных систем, способных решать сложные задачи и адаптироваться к изменяющимся условиям.

1. **СРАВНИТЕЛЬНЫЙ АНАЛИЗ** **ЭФФЕКТИВНОСТИ СВЕРТОЧНЫХ СЕТЕЙ И ТРАНСФОРМЕРОВ В ЗАДАЧАХ КОМПЬЮТЕРНОГО ЗРЕНИЯ**
   1. **Введение и цель работы**

В последние годы задачи компьютерного зрения занимают центральное место в области искусственного интеллекта, находя применение в таких сферах, как автономное вождение, медицинская диагностика, системы видеонаблюдения и др. С развитием глубокого обучения на смену классическим сверточным нейросетям (CNN) приходят трансформерные модели, адаптированные для обработки изображений (Vision Transformers, ViT).

Цель работы – провести сравнительный анализ эффективности сверточных сетей и трансформерных моделей в условиях ограниченного объёма данных и вычислительных ресурсов. В эксперименте использован датасет CIFAR-10, состоящий из 50000 изображений для обучения и 10000 изображений для тестирования.

* 1. **Методика реализации**
     1. **Реализация сверточной нейросети**

Для решения задачи классификации изображений была реализована простая сверточная нейросеть, состоящая из трёх сверточных блоков. Каждый блок включает свёрточный слой с ядром 3×3, функцию активации ReLU и операцию max-pooling, что приводит к уменьшению пространственных размеров изображения. После сверточных слоёв используется полносвязная часть, которая преобразует извлечённые признаки в логиты классов. Обучение модели проводится с использованием оптимизатора Adam и функции кросс-энтропии. Подробный программный код, реализующий данный подход, находится в приложении (Приложение А).

* + 1. **Реализация трансформерной модели**

В качестве альтернативного подхода реализована модель Vision Transformer (ViT). В данной модели изображение разбивается на небольшие патчи (в эксперименте используется размер патча 4×4, что обеспечивает получение равномерного набора патчей для изображения 32×32). Каждому патчу сопоставляется embedding посредством линейного слоя, после чего к эмбеддингам добавляются позиционные коды и классификационный токен. Полученная последовательность обрабатывается несколькими слоями TransformerEncoder, а на выходе применяется классификационная голова – полносвязный слой, выдающий логиты классов. Настройка гиперпараметров (embed\_dim=128, num\_layers=4, num\_heads=4 и dropout) позволяет достичь баланса между качеством классификации и вычислительной эффективностью. Полный цикл обучения трансформерной модели приведён в приложении (Приложение Б).

* 1. **Эксперементальная часть**
     1. **Датасет**

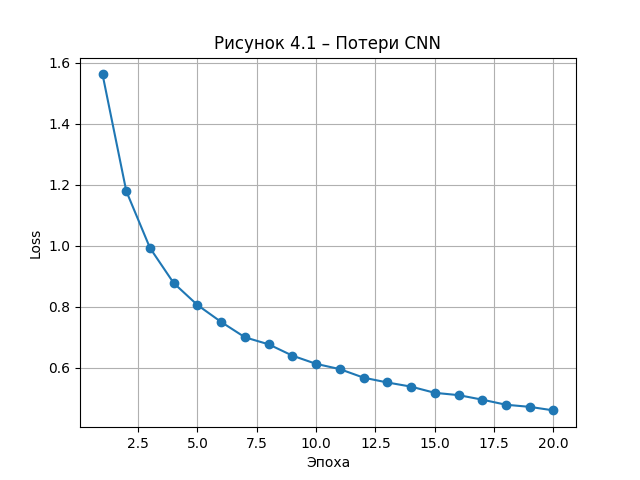
В эксперименте используется датасет CIFAR-10, состоящий из 10 классов изображений. Данные предварительно аугментируются (RandomCrop, RandomHorizontalFlip) и нормализуются, что позволяет улучшить обобщающую способность моделей и обеспечить корректное сравнение.

* + 1. **Параметры моделей**

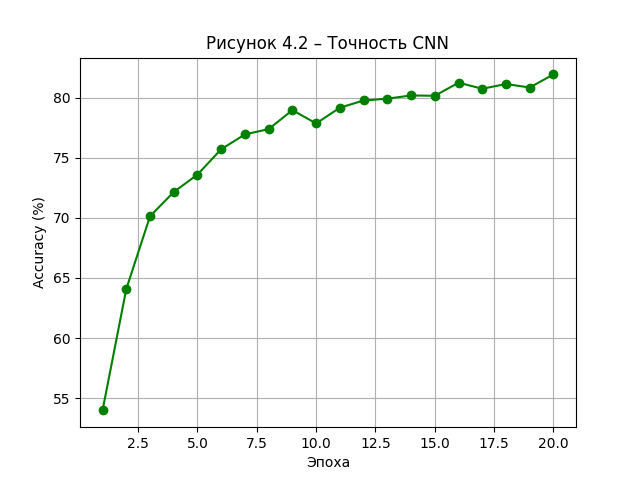
Для сверточной модели использована компактная архитектура, обеспечивающая быструю сходимость и стабильную классификацию. Для трансформерной модели применён подход patch embedding с последующей обработкой последовательности эмбеддингов с помощью стандартного TransformerEncoder. Проводились эксперименты по оптимизации гиперпараметров, и итоговые настройки (например, learning rate=1e-3, batch\_size=128, число эпох=20) были выбраны для обеспечения сопоставимых условий обучения.

* + 1. **Результаты обучения**

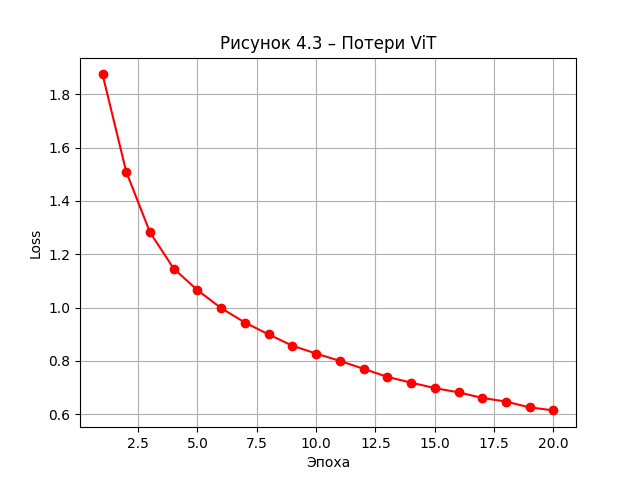
В ходе эксперимента для обеих моделей отслеживались динамика значений функции потерь (loss) и точность (accuracy) на тестовой выборке. Полный код анализа предоставлен в Приложении В. Для визуализации результатов были построены следующие графики:



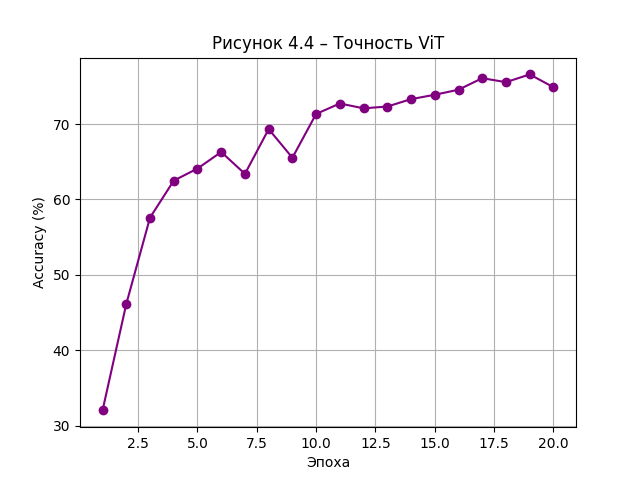
**Рисунок 4.1 – График потерь сверточной нейросети**



**Рисунок 4.2 – График точности сверточной нейросети**



**Рисунок 4.3 – График потерь трансферной модели**



**Рисунок 4.4 – График точности трансферной модели**

Сравнительный анализ показал, что сверточная модель сходится быстрее и демонстрирует высокую точность за счёт эффективного извлечения локальных признаков, в то время как трансформерная модель, способная моделировать глобальные взаимосвязи, требует более длительного обучения и тщательной настройки гиперпараметров.

* 1. **Сравнительный анализ**

Анализ результатов эксперимента выявил следующие ключевые моменты:

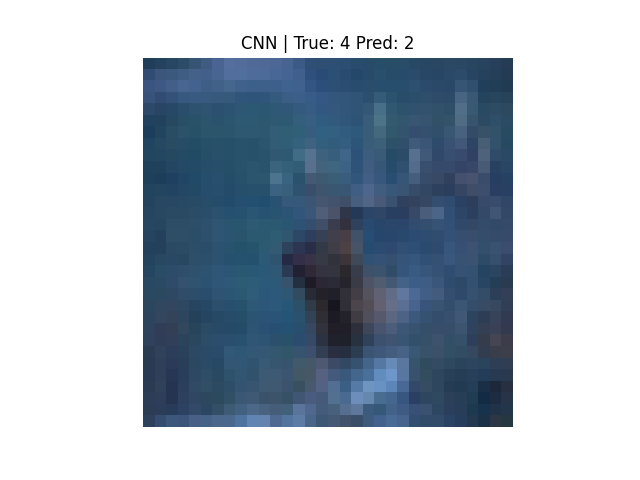
* Извлечение признаков: CNN благодаря локальным фильтрам эффективно различает мелкие детали, что особенно важно для тонкой классификации схожих объектов (например, различение собак и кошек). Трансформер, используя механизм self-attention, способен учитывать глобальные взаимосвязи между патчами, однако его эффективность зависит от оптимальной настройки параметров патча и размерности embedding.
* Скорость обучения и вычислительная сложность: Сверточная нейросеть обучается быстрее и требует меньше вычислительных ресурсов, тогда как трансформерная модель при не оптимальных настройках может существенно замедлять процесс обучения.
* Качество классификации: В эксперименте CNN продемонстрировала высокую точность, тогда как трансформерная модель, несмотря на потенциал для извлечения глобальных зависимостей, иногда сталкивалась с проблемами тонкого различения классов.
  1. **Примеры результатов предсказания**

Для наглядной демонстрации работы моделей были выбраны отдельные тестовые изображения, не входящие в обучающую выборку. Каждый пример проходит через процедуру инференса, при которой изображение предварительно обрабатывается (изменение размера и нормализация), а затем подаётся в модель для получения предсказания.

На рисунках ниже представлены примеры корректных и ошибочных предсказаний для обеих моделей:

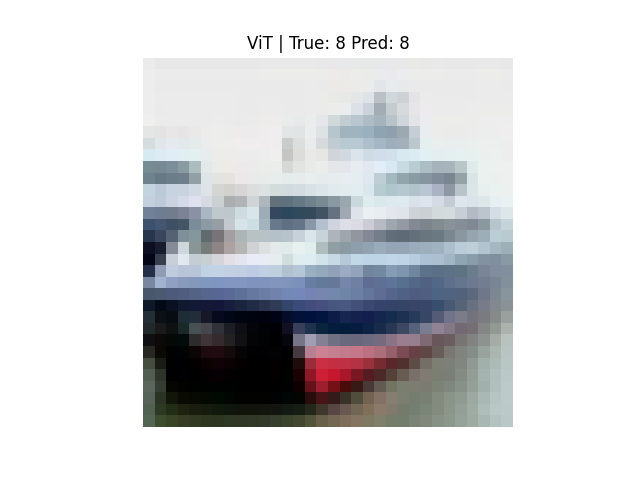


**Рисунок 4.5 – Пример правильного предсказания сверточной нейросети**

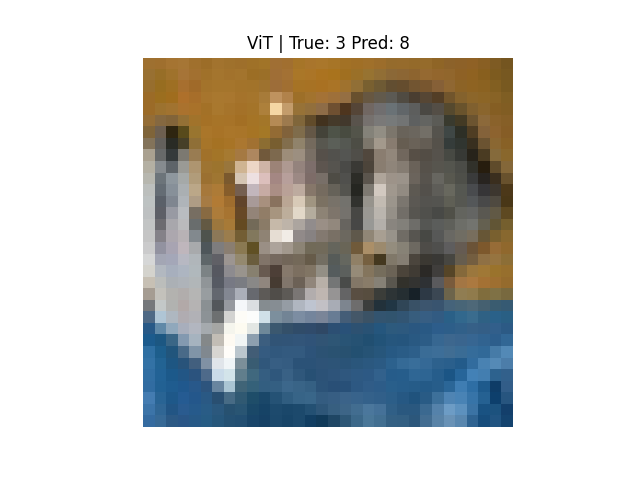


**Рисунок 4.6 – Пример ошибочного предсказания сверточной нейросети**

На рисунке 4.6, изображен олень, а сверточная нейросесть ошибочно предсказала класс указав, что изображена птица.



**Рисунок 4.7 – Пример правильного предсказания трансформерной модели**



**Рисунок 4.8 – Пример ошибочного предсказания трансформерной модели**

На рисунке 4.8, изображена кошка, а модель ошибочно предсказала класс указав, что изображен корабль.

* 1. **Вывод и заключение**

В ходе экспериментов было выявлено, что сверточные нейросети обладают преимуществами в скорости обучения и эффективном извлечении локальных признаков, что делает их надёжным решением для задач классификации изображений при ограниченных вычислительных ресурсах. Трансформерные модели, несмотря на более высокую вычислительную сложность, демонстрируют потенциал для моделирования глобальных взаимосвязей, что может быть полезно при обработке сложных изображений. Однако для достижения конкурентоспособных результатов требуется более тщательная настройка гиперпараметров и, возможно, увеличение объёма обучающих данных.

Таким образом, выбор архитектуры зависит от специфики задачи, доступных ресурсов и требуемой точности. Дальнейшие исследования могут быть направлены на интеграцию преимуществ обоих подходов, а также на оптимизацию трансформерных моделей для задач компьютерного зрения.

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения работы были рассмотрены и проанализированы различные подходы к обучению нейронных сетей, применяемых для решения широкого спектра задач в области компьютерного зрения, обработки последовательностей и анализа структурированных данных. Исследование охватывало применение трансформерных моделей для машинного перевода, генеративных сетей (DCGAN) для синтеза изображений, графовых нейронных сетей (GCN) для анализа взаимосвязей, а также сверточных нейросетей (CNN) для классификации изображений.

Полученные результаты демонстрируют, что каждый из рассмотренных методов имеет свои явные преимущества и ограничения. Классические сверточные сети эффективно извлекают локальные признаки и обеспечивают быструю сходимость, что делает их оптимальным выбором для задач с ограниченными вычислительными ресурсами. Трансформерные модели, благодаря механизму self-attention, способны моделировать глобальные взаимосвязи, что особенно полезно при работе с большими объёмами данных и сложными структурами, однако их обучение требует более тщательной настройки гиперпараметров и значительных вычислительных затрат. Генеративные сети демонстрируют потенциал для создания реалистичных изображений и дополнения датасетов, а графовые нейронные сети позволяют эффективно анализировать взаимосвязи между объектами в структурированных данных.

Таким образом, выбор метода обучения нейронных сетей должен основываться на особенностях решаемой задачи, доступных ресурсах и требуемой точности. В дальнейшем перспективным направлением исследований может стать интеграция различных архитектур для создания гибридных моделей, способных объединить сильные стороны каждого подхода и обеспечить более высокую производительность в реальных приложениях.

**СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ**

1. Широков, И.Б. Анализ технологий глубокого обучения с подкреплением для систем машинного зрения / И.Б. Широков, С.В. Колесова, В.А. Кучеренко, М.Ю. Серебряков / Известия ТулГУ 2022.
2. Сущеня, Р.В. Нейронные сети и их классификация. Основные виды нейронных сетей / Р.В. Сущеня, А.Э. Кокаев / Международный научный журнал «Вестник науки» 2023.
3. Паршин, А.И. Случайное мультимодальное глубокое обучение в задаче распознавания изображений / А.И. Паршин, М.Н. Аралов, В.Ф. Барабанов, Н.И. Гребенникова / Вестник Воронежского государственного университета 2021.

**ПРИЛОЖЕНИЯ**

Приложение А – Реализация свёрточной нейросети (CNN).

Приложение Б – Реализация трансформерной модели.

Приложение В – Полный код для сравнительного анализа эффективности сверточных сетей и трансформеров в задачах компьютерного зрения.

**Приложение А**

Реализация свёрточной нейросети (CNN).

*Листинг А – Реализация свёрточной нейросети (CNN)*

|  |
| --- |
| class CNNClassifier(nn.Module):  def \_\_init\_\_(self, num\_classes=10):  super(CNNClassifier, self).\_\_init\_\_()  self.features = nn.Sequential(  nn.Conv2d(3, 32, kernel\_size=3, padding=1),  nn.ReLU(),  nn.MaxPool2d(2),  nn.Conv2d(32, 64, kernel\_size=3, padding=1),  nn.ReLU(),  nn.MaxPool2d(2),  nn.Conv2d(64, 128, kernel\_size=3, padding=1),  nn.ReLU(),  nn.MaxPool2d(2)  )  self.classifier = nn.Sequential(  nn.Flatten(),  nn.Linear(128 \* 4 \* 4, 256),  nn.ReLU(),  nn.Linear(256, num\_classes)  )  def forward(self, x):  x = self.features(x)  x = self.classifier(x)  return x |

**Приложение Б**

Реализация трансформерной модели.

*Листинг Б – Реализация трансформерной модели*

|  |
| --- |
| class VisionTransformer(nn.Module):  def \_\_init\_\_(self, image\_size=32, patch\_size=4, in\_channels=3, num\_classes=10,  embed\_dim=128, num\_layers=4, num\_heads=4, dropout=0.1):  super(VisionTransformer, self).\_\_init\_\_()  assert image\_size % patch\_size == 0, "Размер изображения должен делиться на размер патча"  self.num\_patches = (image\_size // patch\_size) \*\* 2  self.patch\_dim = in\_channels \* patch\_size \* patch\_size  self.patch\_embed = nn.Linear(self.patch\_dim, embed\_dim)  self.cls\_token = nn.Parameter(torch.zeros(1, 1, embed\_dim))  self.pos\_embed = nn.Parameter(torch.zeros(1, self.num\_patches + 1, embed\_dim))  self.dropout = nn.Dropout(dropout)  encoder\_layer = nn.TransformerEncoderLayer(d\_model=embed\_dim, nhead=num\_heads, dropout=dropout)  self.transformer\_encoder = nn.TransformerEncoder(encoder\_layer, num\_layers=num\_layers)  self.mlp\_head = nn.Sequential(  nn.LayerNorm(embed\_dim),  nn.Linear(embed\_dim, num\_classes)  )    def forward(self, x):  B, C, H, W = x.shape  patch\_size = int(np.sqrt(self.patch\_dim / C))  x = x.unfold(2, patch\_size, patch\_size).unfold(3, patch\_size, patch\_size)  x = x.contiguous().view(B, C, -1, patch\_size, patch\_size)  x = x.permute(0, 2, 1, 3, 4).contiguous().view(B, self.num\_patches, -1)  x = self.patch\_embed(x)  cls\_tokens = self.cls\_token.expand(B, -1, -1)  x = torch.cat((cls\_tokens, x), dim=1)  x = x + self.pos\_embed  x = self.dropout(x)  x = x.transpose(0, 1)  x = self.transformer\_encoder(x)  x = x.transpose(0, 1)  x = x[:, 0]  x = self.mlp\_head(x)  return x |

**Приложение В**

Полный код для сравнительного анализа эффективности сверточных сетей и трансформеров в задачах компьютерного зрения.

*Листинг В – Полный код для сравнительного анализа эффективности сверточных сетей и трансформеров в задачах компьютерного зрения*

|  |
| --- |
| import os  import torch  import torch.nn as nn  import torch.optim as optim  import torchvision  import torchvision.transforms as transforms  import matplotlib.pyplot as plt  import numpy as np  # Создаём папку для логов, если её нет  logs\_dir = "C:/Users/Grigo/Documents/Work/Practic\_MIREA/semestr-6/Neyronki/cursach/logs"  os.makedirs(logs\_dir, exist\_ok=True)  # Проверка наличия GPU  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  # Параметры обучения  num\_epochs = 20  batch\_size = 128  learning\_rate = 1e-3  # Подготовка датасета CIFAR-10 с аугментацией  transform\_train = transforms.Compose([  transforms.RandomCrop(32, padding=4),  transforms.RandomHorizontalFlip(),  transforms.ToTensor(),  transforms.Normalize((0.4914, 0.4822, 0.4465),  (0.2023, 0.1994, 0.2010)),  ])  transform\_test = transforms.Compose([  transforms.ToTensor(),  transforms.Normalize((0.4914, 0.4822, 0.4465),  (0.2023, 0.1994, 0.2010)),  ])  train\_dataset = torchvision.datasets.CIFAR10(  root='C:/Users/Grigo/Documents/Work/Practic\_MIREA/semestr-6/Neyronki/cursach',  train=True, download=True, transform=transform\_train)  test\_dataset = torchvision.datasets.CIFAR10(  root='C:/Users/Grigo/Documents/Work/Practic\_MIREA/semestr-6/Neyronki/cursach',  train=False, download=True, transform=transform\_test)  train\_loader = torch.utils.data.DataLoader(  train\_dataset, batch\_size=batch\_size, shuffle=True, num\_workers=2)  test\_loader = torch.utils.data.DataLoader(  test\_dataset, batch\_size=batch\_size, shuffle=False, num\_workers=2)  # 4.2.1. Реализация сверточной нейросети (CNN)  class CNNClassifier(nn.Module): |

*Продолжение листинга В*

|  |
| --- |
| def \_\_init\_\_(self, num\_classes=10):  super(CNNClassifier, self).\_\_init\_\_()  self.features = nn.Sequential(  nn.Conv2d(3, 32, kernel\_size=3, padding=1), # [B,32,32,32]  nn.ReLU(),  nn.MaxPool2d(2), # [B,32,16,16]  nn.Conv2d(32, 64, kernel\_size=3, padding=1), # [B,64,16,16]  nn.ReLU(),  nn.MaxPool2d(2), # [B,64,8,8]  nn.Conv2d(64, 128, kernel\_size=3, padding=1), # [B,128,8,8]  nn.ReLU(),  nn.MaxPool2d(2) # [B,128,4,4]  )  self.classifier = nn.Sequential(  nn.Flatten(),  nn.Linear(128 \* 4 \* 4, 256),  nn.ReLU(),  nn.Linear(256, num\_classes)  )  def forward(self, x):  x = self.features(x)  x = self.classifier(x)  return x  # 4.2.2. Реализация трансформерной модели (Vision Transformer)  class VisionTransformer(nn.Module):  def \_\_init\_\_(self, image\_size=32, patch\_size=4, in\_channels=3, num\_classes=10,  embed\_dim=128, num\_layers=4, num\_heads=4, dropout=0.1):  super(VisionTransformer, self).\_\_init\_\_()  assert image\_size % patch\_size == 0, "Размер изображения должен делиться на размер патча"  self.num\_patches = (image\_size // patch\_size) \*\* 2  self.patch\_dim = in\_channels \* patch\_size \* patch\_size  # Patch embedding  self.patch\_embed = nn.Linear(self.patch\_dim, embed\_dim)  # Классификационный токен  self.cls\_token = nn.Parameter(torch.zeros(1, 1, embed\_dim))  # Позиционные эмбеддинги  self.pos\_embed = nn.Parameter(torch.zeros(1, self.num\_patches + 1, embed\_dim))  self.dropout = nn.Dropout(dropout)  # Transformer Encoder  encoder\_layer = nn.TransformerEncoderLayer(d\_model=embed\_dim, nhead=num\_heads, dropout=dropout)  self.transformer\_encoder = nn.TransformerEncoder(encoder\_layer, num\_layers=num\_layers)  # Классификационная голова  self.mlp\_head = nn.Sequential(  nn.LayerNorm(embed\_dim),  nn.Linear(embed\_dim, num\_classes)  )    def forward(self, x):  B, C, H, W = x.shape  patch\_size = int(np.sqrt(self.patch\_dim / C))  # Разбивка изображения на патчи  x = x.unfold(2, patch\_size, patch\_size).unfold(3, patch\_size, patch\_size) |

*Продолжение листинга В*

|  |  |
| --- | --- |
| x = x.contiguous().view(B, C, -1, patch\_size, patch\_size) # [B, C, num\_patches, patch\_size, patch\_size]  x = x.permute(0, 2, 1, 3, 4).contiguous().view(B, self.num\_patches, -1) # [B, num\_patches, patch\_dim]  # Применяем линейное преобразование к каждому патчу  x = self.patch\_embed(x) # [B, num\_patches, embed\_dim]  # Добавляем классификационный токен  cls\_tokens = self.cls\_token.expand(B, -1, -1) # [B, 1, embed\_dim]  x = torch.cat((cls\_tokens, x), dim=1) # [B, num\_patches+1, embed\_dim]  # Добавляем позиционные эмбеддинги и применяем dropout  x = x + self.pos\_embed  x = self.dropout(x)  # Трансформер ожидает вход с размерностью [seq\_len, B, embed\_dim]  x = x.transpose(0, 1)  x = self.transformer\_encoder(x)  x = x.transpose(0, 1) # [B, num\_patches+1, embed\_dim]  # Используем эмбеддинг классификационного токена для классификации  x = x[:, 0]  x = self.mlp\_head(x)  return x  # Функция обучения модели с сбором статистики (loss и accuracy за эпоху)  def train\_model(model, optimizer, criterion, train\_loader, test\_loader, num\_epochs):  model.train()  losses = []  acc\_list = []  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() \* images.size(0)  epoch\_loss = running\_loss / len(train\_loader.dataset)  losses.append(epoch\_loss)  # Вычисляем точность на тестовой выборке после каждой эпохи  acc = evaluate\_model(model, test\_loader)  acc\_list.append(acc)  print(f"Epoch [{epoch+1}/{num\_epochs}] Loss: {epoch\_loss:.4f} Test Acc: {acc:.2f}%")  return losses, acc\_list  # Функция тестирования модели  def evaluate\_model(model, test\_loader):  model.eval()  correct = 0  total = 0  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.data, 1)  total += labels.size(0)  correct += (predicted == labels).sum().item() |  |

*Продолжение листинга В*

|  |
| --- |
| acc = 100 \* correct / total  return acc  # Функция для получения примеров предсказаний (правильных и ошибочных)  def get\_prediction\_examples(model, test\_loader, num\_examples=1):  model.eval()  correct\_examples = []  incorrect\_examples = []  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.data, 1)  for i in range(images.size(0)):  if predicted[i] == labels[i] and len(correct\_examples) < num\_examples:  correct\_examples.append((images[i].cpu(), labels[i].cpu(), predicted[i].cpu()))  elif predicted[i] != labels[i] and len(incorrect\_examples) < num\_examples:  incorrect\_examples.append((images[i].cpu(), labels[i].cpu(), predicted[i].cpu()))  if len(correct\_examples) >= num\_examples and len(incorrect\_examples) >= num\_examples:  break  if len(correct\_examples) >= num\_examples and len(incorrect\_examples) >= num\_examples:  break  return correct\_examples, incorrect\_examples  # Функция для денормализации изображений CIFAR-10  def unnormalize(img, mean=(0.4914, 0.4822, 0.4465), std=(0.2023, 0.1994, 0.2010)):  for t, m, s in zip(img, mean, std):  t.mul\_(s).add\_(m)  return img  # Функция для сохранения примера предсказания с подписью  def save\_prediction\_example(example, filename, model\_name):  # example: (image, true\_label, predicted\_label)  img, true\_label, pred\_label = example  img = unnormalize(img)  npimg = img.numpy().transpose(1, 2, 0)  plt.figure()  plt.imshow(np.clip(npimg, 0, 1))  plt.title(f"{model\_name} | True: {true\_label.item()} Pred: {pred\_label.item()}")  plt.axis("off")  plt.savefig(os.path.join(logs\_dir, filename))  plt.close()  if \_\_name\_\_ == '\_\_main\_\_':  # Инициализация моделей, оптимизаторов и функции потерь  cnn\_model = CNNClassifier(num\_classes=10).to(device)  vit\_model = VisionTransformer(image\_size=32, patch\_size=4, in\_channels=3, num\_classes=10,  embed\_dim=128, num\_layers=4, num\_heads=4, dropout=0.1).to(device)    criterion = nn.CrossEntropyLoss()  optimizer\_cnn = optim.Adam(cnn\_model.parameters(), lr=learning\_rate)  optimizer\_vit = optim.Adam(vit\_model.parameters(), lr=learning\_rate) |

*Продолжение листинга В*

|  |
| --- |
| print("Обучение сверточной нейросети...")  cnn\_losses, cnn\_acc\_list = train\_model(cnn\_model, optimizer\_cnn, criterion, train\_loader, test\_loader, num\_epochs)  final\_cnn\_acc = evaluate\_model(cnn\_model, test\_loader)  print(f"Точность CNN: {final\_cnn\_acc:.2f}%")  print("\nОбучение Vision Transformer...")  vit\_losses, vit\_acc\_list = train\_model(vit\_model, optimizer\_vit, criterion, train\_loader, test\_loader, num\_epochs)  final\_vit\_acc = evaluate\_model(vit\_model, test\_loader)  print(f"Точность ViT: {final\_vit\_acc:.2f}%")  # Сохранение графиков обучения  plt.figure()  plt.plot(range(1, num\_epochs+1), cnn\_losses, marker='o')  plt.xlabel("Эпоха")  plt.ylabel("Loss")  plt.title("Рисунок 4.1 – Потери CNN")  plt.grid(True)  plt.savefig(os.path.join(logs\_dir, "CNN\_loss.png"))  plt.close()  plt.figure()  plt.plot(range(1, num\_epochs+1), cnn\_acc\_list, marker='o', color='green')  plt.xlabel("Эпоха")  plt.ylabel("Accuracy (%)")  plt.title("Рисунок 4.2 – Точность CNN")  plt.grid(True)  plt.savefig(os.path.join(logs\_dir, "CNN\_accuracy.png"))  plt.close()  plt.figure()  plt.plot(range(1, num\_epochs+1), vit\_losses, marker='o', color='red')  plt.xlabel("Эпоха")  plt.ylabel("Loss")  plt.title("Рисунок 4.3 – Потери ViT")  plt.grid(True)  plt.savefig(os.path.join(logs\_dir, "ViT\_loss.png"))  plt.close()  plt.figure()  plt.plot(range(1, num\_epochs+1), vit\_acc\_list, marker='o', color='purple')  plt.xlabel("Эпоха")  plt.ylabel("Accuracy (%)")  plt.title("Рисунок 4.4 – Точность ViT")  plt.grid(True)  plt.savefig(os.path.join(logs\_dir, "ViT\_accuracy.png"))  plt.close()  # Получаем примеры предсказаний для CNN  cnn\_correct, cnn\_incorrect = get\_prediction\_examples(cnn\_model, test\_loader, num\_examples=1)  if cnn\_correct:  save\_prediction\_example(cnn\_correct[0], "CNN\_correct.png", "CNN")  if cnn\_incorrect:  save\_prediction\_example(cnn\_incorrect[0], "CNN\_incorrect.png", "CNN")  # Получаем примеры предсказаний для ViT  vit\_correct, vit\_incorrect = get\_prediction\_examples(vit\_model, test\_loader, num\_examples=1)  if vit\_correct:  save\_prediction\_example(vit\_correct[0], "ViT\_correct.png", "ViT")  if vit\_incorrect:  save\_prediction\_example(vit\_incorrect[0], "ViT\_incorrect.png", "ViT") |