ПРАКТИЧЕСКАЯ РАБОТА №3: СВЕРТОЧНАЯ НЕЙРОННАЯ СЕТЬ

Цель

Научиться проектировать сверточные нейронные сети, понимать их архитектуру и обучать их на задачах классификации изображений. Освоить использование свёрток, пулинга и других ключевых операций.

Задание

Часть 1: Общий пример (1 пара)

1. Загрузка и исследование данных:
- Загрузите датасет CIFAR-10 или аналогичный из библиотеки keras.datasets или torchvision.datasets.
- Выполните предварительную обработку данных: нормализация изображений и преобразование классов в категориальный вид.

2. Создание сверточной нейронной сети:
- Архитектура сети:
- Первый сверточный слой с фильтрами (3x3), активация ReLU.
- Пулинг-слой (2x2).
- Добавьте 2-3 сверточных слоя с последующими пулингами.
- Полносвязный слой с Softmax для классификации.
- Инициализация модели с использованием TensorFlow или PyTorch.

3. Обучение сети:
- Оптимизатор: Adam.

- Функция потерь: кросс-энтропия.
- Обучите сеть на 10-20 эпохах и сохраните результаты.

4. Визуализация:

- Графики изменения потерь и точности на обучающей и тестовой выборках.
- Визуализация фильтров первого сверточного слоя.

Часть 2: Индивидуальные задания (2 пара)

1. Параметры сети:

- Каждому студенту предоставляется уникальный набор параметров:
- Количество фильтров (32, 64, 128).
- Размеры фильтров (3x3, 5x5).
- Функции активации (ReLU, Sigmoid).
- Постройте сверточную сеть с предложенными параметрами и обучите её на заданном датасете.

2. Анализ моделей:

- Сравните результаты обучения при изменении параметров сети (фильтры, размер свёрток).
- Проанализируйте, как параметры влияют на переобучение и точность на тестовой выборке.

Часть 3: Защита и интерпретация результатов (3 пара)

1. Защитите свою модель, представив основные результаты:

- Итоговые метрики (точность, F1-мера).
- Влияние архитектурных параметров на обучение и переобучение.

2. Используйте инструменты интерпретации (например, Grad-CAM) для

визуализации важных областей изображений, на которые обращала внимание сеть.

Что нужно вставить в отчет

1. Введение:
- Описание целей работы и задачи классификации.
- Краткое описание архитектуры сверточной сети и её компонентов.

2. Описание архитектуры:
- Схема архитектуры сети (количество фильтров, размеры слоев сверток и пулинга).
- Параметры обучения: количество эпох, learning rate, оптимизатор.

3. Визуализация результатов:
- Графики потерь (loss) на обучающей и тестовой выборках.
- Графики точности на обучении и тестировании.
- Визуализация фильтров первого сверточного слоя.

4. Сравнительный анализ:
- Влияние различных параметров сети на качество обучения.
- Сравнительная таблица результатов по вариациям архитектуры.
- Графики переобучения (если оно наблюдается).

5. Интерпретация модели:
- Визуализация важных областей изображения с использованием Grad-CAM или других методов.
- Интерпретация: на что обращала внимание сеть при классификации.

6. Заключение:
- Основные выводы об архитектуре сверточных сетей.
- Рекомендации по выбору параметров сети.
- Проблемы, возникшие при обучении, и их решения.

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ ПО 3 ПРАКТИЧЕСКОЙ

Теоретический материал по 1 части

Сверточная нейронная сеть — это разновидность глубокой нейронной сети, разработанная специально для обработки данных с сеточной структурой (например, изображений).

Ключевая идея CNN — автоматическое извлечение признаков из изображений через специальные сверточные слои вместо ручного извлечения признаков.

Основные компоненты CNN:

1. Сверточный слой (Convolutional Layer)

- Использует ядра (фильтры), которые «сканируют» изображение и вычисляют фильтрованные карты признаков (feature maps).
- Позволяет извлекать локальные шаблоны, такие как границы, текстуры и формы.
- Основные параметры:
- Размер фильтра (обычно 3x3, 5x5)
- Количество фильтров
- Шаг (stride)
- Заполнение (padding)

2. Активационная функция (ReLU)

- После свёртки обычно применяется функция активации ReLU (f(x) = max(0, x)), чтобы ввести нелинейность.

3. Слой пулинга (Pooling Layer)

- Выполняет субдискретизацию (уменьшение размера карт признаков).

- Самый распространённый вид — MaxPooling, который берёт максимум на каждом участке (обычно 2x2).
- Уменьшает размерность, снижает переобучение и ускоряет обучение.

4. Полносвязный слой (Fully Connected Layer)

- Завершающая часть сети, где признаки преобразуются в вероятности классов.
- Последний слой — Softmax, преобразующий выход в вероятностное распределение по классам.

Загрузка датасета

Для задач классификации изображений часто используются **открытые датасеты**, предоставляемые библиотеками глубокого обучения. Один из наиболее популярных — **CIFAR-10**.

Что такое CIFAR-10:

- **C**anadian **I**nstitute **F**or **A**dvanced **R**esearch
- Набор данных из **60 000 цветных изображений** (размер 32x32 пикселя)
- Разбит на:
  - 50 000 изображений для обучения
  - **10 000 изображений** для тестирования
  - **10 классов**: самолет, автомобиль, птица, кошка, олень, собака, лягушка, лошадь, корабль, грузовик

Как загрузить?

С помощью популярных библиотек Keras или же PyTorch:

1 вариант Keras

```
from tensorflow.keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

2 вариант PyTorch

```
import torchvision
import torchvision.transforms as transforms
transform = transforms.ToTensor()
trainset = torchvision.datasets.CIFAR10(root='./data',
train=True, download=True, transform=transform)
testset = torchvision.datasets.CIFAR10(root='./data',
train=False, download=True, transform=transform)
```

Исследование структуры данных

Перед началом обучения полезно исследовать форму и тип загруженных данных.

- Размер обучающей выборки: x_train.shape → (50000, 32, 32, 3)
- Размер тестовой выборки: x_test.shape → (10000, 32, 32, 3)
- Тип меток: массив целых чисел от 0 до 9 (y_train, y_test)

Для визуального анализа можно отобразить несколько изображений с их метками.

```
import matplotlib.pyplot as plt
plt.imshow(x_train[0])
plt.title(f"Class: {y_train[0]}")
```

Предварительная обработка данных

Перед обучением необходимо выполнить следующие действия:

а) Нормализация изображений

Пиксели изображений изначально находятся в диапазоне от 0 до 255. Для ускорения и стабилизации обучения значения следует нормализовать до диапазона [0, 1].

```
x_train = x_train / 255.0
x_test = x_test / 255.0
```

В PyTorch нормализация часто реализуется через transforms.Normalize.

б) Преобразование меток классов

Для обучения модели с функцией потерь categorical_crossentropy метки необходимо преобразовать в категориальный (one-hot) формат.

```
from tensorflow.keras.utils import to_categorical
y_train_cat = to_categorical(y_train, num_classes=10)
y_test_cat = to_categorical(y_test, num_classes=10)
```

Для PyTorch такое преобразование не требуется, если используется nn.CrossEntropyLoss, так как она принимает целочисленные метки.

Создание сверточной нейронной сети

Общая структура сети

Сверточная нейронная сеть (CNN) представляет собой последовательность чередующихся сверточных и пулинг-слоев, за которыми следуют полносвязные слои. На выходе сеть выдает вектор вероятностей принадлежности к классам.

Базовая архитектура может включать:

• Входной слой: принимает изображения размера 32×32×3

• Сверточные слои (Conv2D): извлекают признаки с помощью фильтров

• Пулинг-слои (MaxPooling2D): уменьшают размерность признаков

• Полносвязные (Dense) слои: преобразуют признаки в классификационное решение

• Softmax: используется на выходе для получения вероятностей по классам

Конкретная архитектура для задания

Первый сверточный слой:

• Размер фильтра: 3×3

• Активация: ReLU

• Количество фильтров: 32 или 64

Пулинг: MaxPooling 2×2

Дополнительные сверточные + пулинг-слои (2–3 блока)

• С увеличением числа фильтров (например, 64, 128)

Переход к полносвязной части:

• Flatten: преобразует тензор в вектор

• Dense: скрытый слой (например, 128 нейронов)

• Dense: выходной слой с Softmax на 10 классов

Реализация в TensorFlow (пример)

```
from tensorflow.keras import models, layers
model = models.Sequential()
model.add(layers.Conv2D(32,    (3,    3),    activation='relu',
input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Реализация в PyTorch (пример)

```
import torch.nn as nn
import torch.nn.functional as F

class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3)
        self.fc1 = nn.Linear(128 * 2 * 2, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
```

```
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = F.relu(self.conv3(x))
        x = x.view(-1, 128 * 2 * 2)
        x = F.relu(self.fc1(x))
        x = F.softmax(self.fc2(x), dim=1)
        return x
```

Обучение сети

Оптимизатор: Adam

Adam (Adaptive Moment Estimation) — оптимизатор, сочетающий идеи RMSProp и Momentum. Он автоматически регулирует скорость обучения для каждого параметра.

Преимущества:

• Быстрая сходимость

• Хорошо работает “из коробки”

Функция потерь: кросс-энтропия

Функция потерь измеряет расхождение между предсказанным распределением (Softmax) и истинной меткой.

Для многоклассовой классификации используется **categorical crossentropy**

Процесс обучения

- Количество эпох: 10–20
- Размер батча: 32–128

- На каждой эпохе сеть проходит через всю обучающую выборку
- Параллельно отслеживаются метрики на тестовой выборке

Keras

```
model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train,            y_train_cat,            epochs=15,
validation_data=(x_test, y_test_cat))
```

Визуализация результатов

Графики обучения

Позволяют визуально оценить, насколько хорошо обучается сеть:

- Потери (loss) на обучении и валидации
- Точность (accuracy) на обучении и валидации

Keras

```
import matplotlib.pyplot as plt

history = model.fit(...)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
```

Визуализация фильтров

Фильтры первого сверточного слоя часто можно интерпретировать визуально. Они показывают, какие шаблоны (границы, цветовые переходы и т.п.) извлекает модель на ранней стадии.

Пример для Keras:

```
weights = model.layers[0].get_weights()[0]  # shape: (3, 3, 3,
32)
```

Теоретический материал по 2 части

Цель этой части — изучить, как различные архитектурные параметры влияют на качество обучения сверточной нейронной сети, её обобщающую способность и устойчивость к переобучению. Каждому студенту предоставляется индивидуальный набор параметров, на основе которого он проектирует и обучает модель.

Количество фильтров

Фильтры в сверточных слоях извлекают признаки различной степени абстракции. Количество фильтров определяет “глубину” карты признаков.

- Меньшее число фильтров (32):

Меньше параметров, быстрее обучение, риск недообучения.

- Большее число фильтров (64, 128):

Способны извлекать больше признаков, но увеличивают вычислительную нагрузку и могут привести к переобучению при недостатке данных.

Размеры фильтров

Размер фильтра влияет на область восприятия сверточного слоя:

- 3×3 — захватывает локальные шаблоны, более точная обработка текстур, применяется чаще всего.
- 5×5 — покрывает более широкую область, но требует больше параметров и вычислений. Может приводить к потере мелких деталей.

На практике лучше использовать несколько последовательных 3×3

фильтров, чем один 5×5.

Функции активации

Активационные функции вводят нелинейность, позволяя сети моделировать сложные зависимости.

- ReLU (Rectified Linear Unit):

Функция вида $f(x) = max(0, x)$. Быстрая, устойчивая, способствует хорошей сходимости.

- Sigmoid:

Функция $f(x) = \frac{1}{1 + e^{-x}}$. Приводит значения в диапазон (0, 1). Склонна к затухающим градиентам и обучается медленнее.

Выбор функции активации влияет на динамику обучения и глубину сети, которую можно эффективно обучить.

Метрики оценки

Для оценки качества модели используются:

- **Точность (accuracy)** — доля правильно классифицированных объектов.
- **Функция потерь (loss)** — отражает расхождение между предсказанными и истинными метками.
- **Дополнительно:** F1-мера, precision, recall, confusion matrix.

Переобучение

Переобучение возникает, когда модель слишком точно запоминает

обучающую выборку и плохо обобщает на тестовые данные.

Признаки переобучения:

- Loss на обучении снижается, но на тесте — растет
- Accuracy на обучении высокая, на тесте — низкая

Причины:

- Слишком сложная архитектура (много фильтров, большие Dense-слои)
- Недостаток данных
- Отсутствие регуляризации (dropout, batch normalization)

Методы борьбы:

- Уменьшение размеров модели
- Добавление Dropout-слоев
- Использование аугментации данных
- Раннее завершение обучения (early stopping)

Сравнительный анализ

Рекомендуется оформить результаты экспериментов в виде таблиц и графиков.

Пример таблицы

Параметры модели	Train Accuracy	Test Accuracy	Loss (val)	Переобучение
32 фильтра, 3×3, ReLU	0.87	0.83	0.46	Нет
128 фильтров, 5×5, Sigmoid	0.98	0.74	1.10	Да

Теоретический материал по 3 части

Завершающий этап работы включает в себя обоснование эффективности обученной модели, интерпретацию её поведения и анализ влияния архитектурных параметров. Студент должен представить основные метрики

качества и провести аналитическую защиту архитектурных решений.

Для комплексной оценки модели необходимо использовать несколько метрик:

- Точность (Accuracy):

Доля правильно классифицированных объектов от общего числа.

- F1-мера:

Гармоническое среднее между точностью (Precision) и полнотой (Recall).

Используется при несбалансированных классах.

- Матрица ошибок (confusion matrix):

Показывает, какие классы чаще всего путает модель.

- Графики потерь и точности:

Отражают динамику обучения и признаки переобучения.

При защите модели следует объяснить:

- Почему были выбраны именно эти значения количества фильтров, размера свёрток и функций активации
- Как изменения этих параметров повлияли на:
- скорость обучения,
- уровень переобучения,
- качество на тестовой выборке
- Какие конфигурации дали наилучший результат и почему

Аргументы должны опираться на эмпирические наблюдения (графики, таблицы, метрики), а не на интуитивные предположения.

Интерпретация модели с использованием Grad-CAM

Современные методы интерпретации позволяют понять, на какие части изображения обращала внимание нейросеть при принятии решения. Один из таких методов — Grad-CAM (Gradient-weighted Class Activation Mapping).

Grad-CAM — это визуальный инструмент, который показывает, какие

области изображения наиболее сильно повлияли на результат классификации.

Принцип работы:

- Вычисляются градиенты выхода выбранного класса по последнему сверточному слою
- Градиенты агрегируются, формируя карту важности
- Карта проецируется обратно на изображение в виде тепловой карты

Базовый пример (TensorFlow + tf-keras-vis)

```
from tf_keras_vis.saliency import Gradcam
from tf_keras_vis.utils.model_modifiers import ReplaceToLinear
from tf_keras_vis.utils.scores import CategoricalScore
gradcam = Gradcam(model, model_modifier=ReplaceToLinear())
score = CategoricalScore([target_class_index])
cam = gradcam(score, seed_input=image_input)
```

Полученная карта Grad-CAM накладывается на исходное изображение в виде цветной маски, чтобы выделить наиболее значимые области.