Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Южно-Уральский государственный университет»

* 1. (национальный исследовательский университет)

Институт естественных и точных наук

Кафедра прикладной математики и программирования

Отчет по лабораторной работе № 4

по дисциплине «Современные нейросетевые технологии»

|  |  |
| --- | --- |
|  |  |
|  | Авторы работы  Студент группы ЕТ-122  \_\_\_\_\_\_\_\_\_/ Постаушкин Ф.Ю.  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_2025 г. |
|  |  |
|  | Руководитель работы,  \_\_\_\_\_\_\_\_\_\_/ Кичеев Д. М.  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_2025 г. |

Челябинск 2025

# Оглавление

1 Постановка задания 3

2 Набор данных 4

3 метрики качества 5

4 программная реализация 6

5 Тестирование на различных конфигурациях 8

# Постановка задания

Задача заключается в бинарной классификации: по входному цветному изображению предсказать метку где

Обучаем модель минимизируя функцию потерь BCE (бинарная кросс-энтропия):

# Набор данных

Использован датасет pizza-not-pizza с Kaggle (https://www.kaggle.com/datasets/carlosrunner/pizza-not-pizza).

|  |  |
| --- | --- |
| **Характеристика** | **Значение** |
| Исходный размер изображений | От 287x287 до 512x512 пикселей |
| Количество изображений | 1966 + 7864 (аугментированные) изображений |
| Размер после обработки | 64x64x3 |
| Тренировочные данные | 6291 изображений |
| Валидационные данные | 1573 изображений |
| Тестовые данные | 1966 изображений |

1. Изображения хранятся в папках:

./data/\*.jpg

1. Каждое изображение цветное изображение в формате JPEG.
2. Загруженные изображения приводятся к размеру 64×64.
3. На вход сети подаётся тензор размера

# метрики качества

Основная метрика Accuracy (доля правильно классифицированных примеров):

Дополнительно считаем Precision, Recall и F1-score:

Где TP True Positive (количество истинно положительных ответов), TN True Negative (количество истинно отрицательных ответов), FP False Positive (количество ложно положительных ответов), FN False Negative (количество ложно отрицательных ответов).

# программная реализация

Листинг кода приведен в моем GitHub https://github.com/Mellochik/mnt-lab3.

Файлы в проекте:

1. practice.ipynb

Jupyter Notebook для исследования данных.

Функциональность:

Загружает датасет с помощью библиотеки kagglehub.

Исследует структуру данных (количество изображений, размеры, распределение классов).

Выводит статистику по изображениям в наборах.

1. dataset.py

Этот модуль определяет класс PizzaDataset, который используется для загрузки и предобработки изображений.

Функциональность:

Загружает изображения из указанной директории.

Преобразует изображения в формат torch.Tensor с размером 64x64.

Определяет метки классов: 0 для "не пицца" и 1 для "пицца".

1. model.py

Содержит реализацию нейронных сетей.

RNN (Рекуррентная нейронная сеть): Модель, состоящая из 2 рекуррентных слоёв, предназначенная для обработки данных, представленных в виде последовательностей. В данном случае в качестве последовательностей используются изображения, преобразованные в одномерные массивы пикселей.

LSTM (Long Short-Term Memory): Архитектура рекуррентной сети с механизмом долгосрочной и краткосрочной памяти, позволяющая эффективно учитывать длительные зависимости в последовательных данных.

GRU (Gated Recurrent Units): Модель на основе управляемых рекуррентных блоков, являющаяся более простой альтернативой LSTM, при этом сохраняющая высокую эффективность при работе с последовательностями.

Функциональность:

Классы RNN, LSTM, GRU создают нейронные сети с соответствующими слоями.

1. train\_test.py

Основной скрипт для обучения, валидации и тестирования модели.

Функциональность:

Загружает данные из директории с использованием PizzaDataset.

Создает модель PizzaCNN с заданными параметрами.

Использует BCELoss как функцию потерь и оптимизатор Adam.

Реализует цикл обучения с сохранением лучшей модели.

Выполняет тестирование на тестовом наборе данных и вычисляет метрики (accuracy, precision, recall, F1-score).

Поддерживает использование GPU (CUDA), если доступно.

# Тестирование на различных конфигурациях

Результаты текущей работы (рекуррентные сети):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **Модель** | **Время обучения и тестирования, с** | **Test Accuracy, %** | **Precision** | **Recall** | **F1** | **Confusion Matrix** |
| 1 | RNN | 284.89 | 90.69 | 0.9140 | 0.8962 | 0.9050 | [[911 82]  [101 872]] |
| 2 | LSTM | 394.37 | 91.66 | 0.9141 | 0.9249 | 0.9194 | [[866 88]  [ 76 936]] |
| 3 | GRU | 488.29 | 94.40 | 0.9479 | 0.9402 | 0.9440 | [[928 51]  [ 59 928]] |

Модели из прошлой лабораторной работы и их результаты:

|  |  |
| --- | --- |
| **№** | **Скрытые слои** |
| 1 | (0): Conv2d(3, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  (1): ReLU()  (2): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  (3): Conv2d(32, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  (4): ReLU()  (5): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  (6): Flatten(start\_dim=1, end\_dim=-1)  (7): Linear(in\_features=16384, out\_features=64, bias=True)  (8): ReLU()  (9): Linear(in\_features=64, out\_features=1, bias=True) |
| 2 | (0): Conv2d(3, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  (1): ReLU()  (2): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  (3): Conv2d(32, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  (4): ReLU()  (5): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  (6): Conv2d(64, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  (7): ReLU()  (8): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  (9): Flatten(start\_dim=1, end\_dim=-1)  (10): Linear(in\_features=8192, out\_features=128, bias=True)  (11): ReLU()  (12): Linear(in\_features=128, out\_features=1, bias=True) |
| 3 | (0): Conv2d(3, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  (1): ReLU()  (2): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  (3): Conv2d(32, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  (4): ReLU()  (5): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  (6): Conv2d(64, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  (7): ReLU()  (8): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  (9): Conv2d(128, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  (10): ReLU()  (11): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  (12): Flatten(start\_dim=1, end\_dim=-1)  (13): Linear(in\_features=4096, out\_features=128, bias=True)  (14): ReLU()  (15): Linear(in\_features=128, out\_features=1, bias=True) |
| 4 | (0): Conv2d(3, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  (1): ReLU()  (2): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  (3): Conv2d(32, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  (4): ReLU()  (5): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  (6): Flatten(start\_dim=1, end\_dim=-1)  (7): Linear(in\_features=16384, out\_features=64, bias=True)  (8): ReLU()  (9): Dropout(p=0.5, inplace=False)  (10): Linear(in\_features=64, out\_features=1, bias=True) |

Результаты экспериментов:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Время обучения и тестирования, с** | **Test Accuracy, %** | **Precision** | **Recall** | **F1** | **Confusion Matrix** |
| 1 | 284.89 | 90.69 | 0.9140 | 0.8962 | 0.9050 | [[911 82]  [101 872]] |
| 2 | 394.37 | 91.66 | 0.9141 | 0.9249 | 0.9194 | [[866 88]  [ 76 936]] |
| 3 | 488.29 | 94.40 | 0.9479 | 0.9402 | 0.9440 | [[928 51]  [ 59 928]] |
| 4 | 311.52 | 89.67 | 0.8894 | 0.9083 | 0.8988 | [[862 112]  [ 91 901]] |

Наилучшее качество показала модель под №4. Она имеет количество слоев свертки 4 и на выходном линейном слое размерность 128, что и повышает качество классификации. К этому можно прийти если если посмотреть на первые две модели. Вторая модель имеет на один слой свертки больше, чем первая, но ключевая составляющая как раз-таки выходной слой, он у второй модели больше и поэтому не образуется так называемого «узкого места» в нейросети, в котором модель может переобучиться и начинать «тупить».

Еще интересно посмотреть на модели 1 и 4, они имеют одинаковую архитектуру, но у второй добавляется Dropout слой, который выключает нейроны в случайном порядке. Как видно из результатов – это не прибавило точности, а даже наоборот убавило, так еще и времени затраченное на обучение больше.

Вывод: увеличение глубины сети в сверточных слоях и увеличение размерности выходного линейного слоя, чтобы больше признаков обрабатывалось после свертки.