Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»**
(МИ ВлГУ)

Факультет ИТР

Кафедра ПИн

УТВЕРЖДАЮ

Зав. кафедрой

_____ А.Л. Жизняков
(подпись)

«____» _____ 2024г

БАКАЛАВРСКАЯ РАБОТА

Тема Распознавание человеческой активности на основе канальных матриц и методов искусственного интеллекта

МИВУ.09.03.04–16.000 ВКР

Руководитель

Астафьев А.В.
(фамилия, инициалы)

_____
(подпись) (дата)

Студент ПИн –120
(группа)

Симонова А.М.
(фамилия, инициалы)

_____
(подпись) (дата)

Муром 2024

В работе представлена исследовательская работа по теме «Распознавание человеческой активности на основе канальных матриц и методов искусственного интеллекта». Цель работы – изучение информации о распознание человеческой активности на основе канальных матриц и методов искусственного интеллекта, реализация своего эксперимента в рамках тем, сбор и анализ данных с помощью методов искусственного интеллекта.

Для анализа и обучения данных был выбран язык Python, код реализован в блокноте Google Colab.

В рамках работы было проведено два эксперимента, составлены таблицы и визуализация данных средствами Python.

The paper presents a research paper on the topic "Recognition of human activity based on channel matrices and artificial intelligence methods". The purpose of the work is to study information about the recognition of human activity based on channel matrices and artificial intelligence methods, to implement one's experiment within the framework of topics, to collect and analyze data using artificial intelligence methods.

Python was chosen for data analysis and training, and the code is implemented in Google Colab notepad.

As part of the work, two experiments were conducted, tables were compiled and data visualization using Python tools.

Содержание

Введение

В современном мире, где технологии развиваются с невероятной скоростью, быстрое и надежное распознавание человеческой деятельности в замкнутом пространстве по–прежнему остается открытой проблемой, связанной со многими реальными приложениями, особенно в здравоохранении и биомедицинском мониторинге.

Наиболее используемыми технологиями для задачи распознания человеческой активности являются камеры, датчики движения. Однако за последние годы появилось много исследований, которые предлагают обрабатывать информацию о движение с помощью Wi–fi устройств.

Использование Channel State Information (CSI) сигналов Wi–Fi для распознавания человеческой активности начало развиваться как исследовательское направление примерно в начале 2010–х годов. С тех пор множество исследований было посвящено улучшению точности и надёжности таких систем, а также расширению спектра распознаваемых активностей. Это направление исследований продолжает активно развиваться, предлагая новые возможности для создания интеллектуальных систем, способных взаимодействовать с человеком и его окружением.

В недавних исследованиях изучалось использование антенн Wi–Fi (1D–датчиков) для сегментации тела и обнаружения ключевых точек тела. Группа исследователей добилась высокой эффективности распознания человеческой активности с помощью составленного трехмерного образа также на основе CSI не для одного человека, а сразу для группы людей. Применяемый там инструмент DensePose, позволяющий извлекать 3D–сетчатую модель человеческого тела из двумерных изображений RGB, использовалась ранее только для обработки информации с камер [8].

Актуальность работы является востребованность, поиск альтернативных решений в сфере распознавания человеческой активности.

Целью работы является изучение информации о распознание человеческой активности на основе канальных матриц и методов искусственного интеллекта, реализация своего эксперимента в рамках темы, сбор данных и анализ с помощью методов искусственного интеллекта.

Для выполнения данной цели необходимо проанализировать исследования по теме работы, выявить алгоритм, используемые архитектуры для обучения, провести свой эксперимент, собрав данные, проанализировав и сделать вывод по получившимся результатам.

1. Связанные исследования

В рамках научной конференции «Зворыкинские чтения» была подготовлена и представлена исследовательская часть работы. Для этого были проанализированы статьи, собранные при поддержки другого студента и научного руководителя ВКР.

1.1 Алгоритм распознания человеческой активности в Wi–fi устройстве

Литература показывает, что исследователи использовали две величины, связанные с Wi–Fi, для разработки систем распознавания человеческой активности, а именно уровень принимаемого сигнала (RSSI) и информацию о состоянии канала (CSI). RSSI широко используется при определении выполняемой активности, основанной на наблюдении за изменениями мощности принимаемого сигнала. Основной недостаток RSSI связан с тем, что он измеряет мощность передаваемого сигнала, которая затухает по мере увеличения расстояния между объектом и приемником. Таким образом, чем дальше расстояние между испытуемым и измерительным узлом, тем ниже точность системы.

Значения CSI обеспечивают измерение свойств канала, где на эти свойства сильно влияет окружающая среда [5] и изменения, происходящие в окружающей среде (Рисунок 1, Рисунок 2), такие как движения субъектов, независимо от того, являются ли эти движения такими же маленькими как движение рук или большими, как у субъекта, идущего из одной точки в другую в окружающей среде.

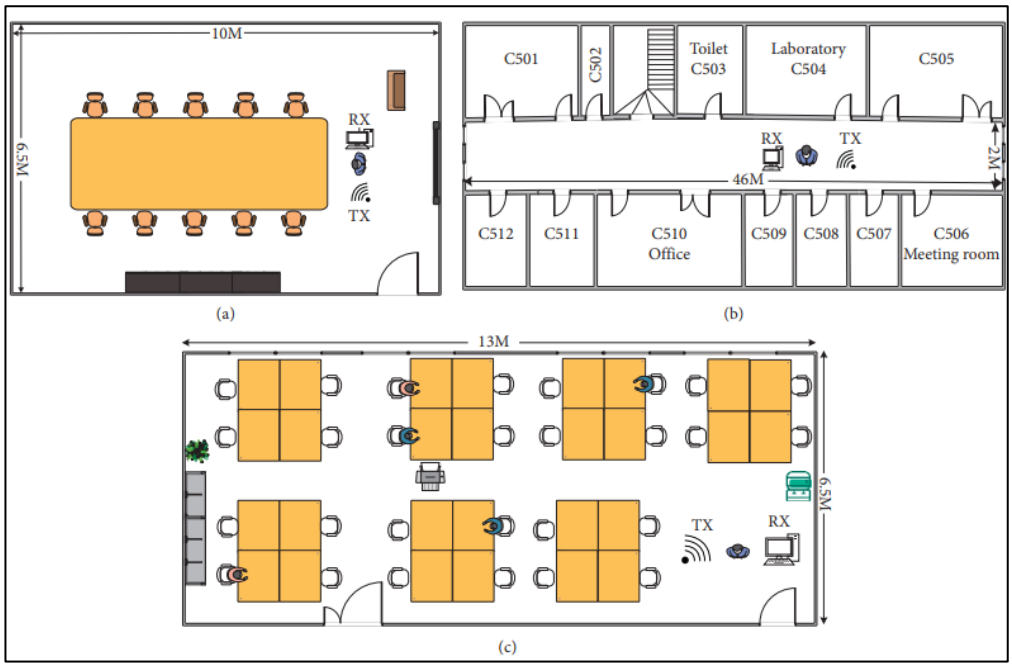

Рисунок 1 – Различные среды

Different scenarios	Detection accuracy of different XingYiQuan actions (%)					
	QiShi	BengQuan	HuQuan	MaXingQuan	ZuanQuan	ShouShi
Meeting room	88.2	89.6	91.0	87.8	87.1	92.3
Corridor	84.3	85.9	86.6	83.6	85.5	88.0
Office	80.9	80.1	81.0	81.0	81.2	83.2

Рисунок 2 – Таблица точности для разных движений в разных средах

В качестве данных используются амплитуда и фаза (по отдельности или вместе). Так как собранный необработанный CSI довольно шумный, его предварительно обрабатывают разными способами [2](Рисунок 3). От устройства с определенным количеством антенн и количества собранных данных зависит результат дальнейшей обработки.

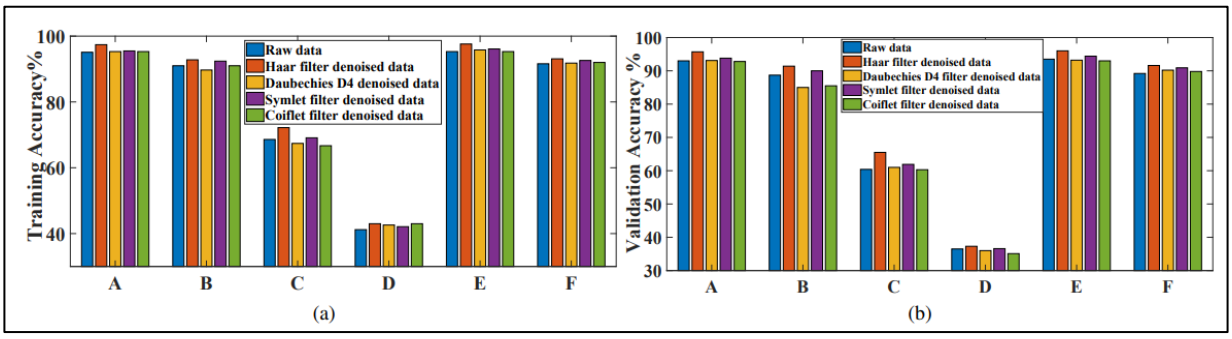

Рисунок 3 – (a) Точность тренировочная % (b) Точность валидационная %

1.2 Сравнение методов классификации

Большинство существующих подходов к распознаванию активности на основе CSI основаны на созданных вручную признаках, которые классифицируются с использованием традиционных методов классификации (например, KNN, SVM и Random Forest), однако последние годы в литературе для этой цели было предложено несколько систем на основе глубокого обучения. Различные исследования сравнивая методы, обращают внимание что алгоритмы глубокого обучения справляются с этой задачей лучше, чаще всего демонстрируя это на примере простой LSTM. Надежность LSTM–подхода была продемонстрирована даже при ухудшении условий эксперимента.

Также вариант использующий одномерный сверточный слой для эффективного извлечения релевантных признаков из входных сигналов CSI и их обработки по всей сети показал себя лучше в сравнение с обычной моделью LSTM [4], при этом сокращая время обучения [6] (Рисунок 4). Иным эффективным подходом оказалась LSTM–система для HAR была представлена в работе [7], где двунаправленная LSTM на основе внимания (BiLSTM) предназначена для изучения признаков с использованием известных последовательностей CSI.

Model	FALL	SIT	SIT-DOWN	STAND	STAND-UP	WALK	ALL	CV
1D-CNN	94.0	98.0	90.0	100.0	95.2	100.0	96.2	94.8
1D-RCNN	89.6	68.0	77.6	92.4	97.6	96.4	86.9	86.8
1D-LSTM	94.8	98.0	93.6	100.0	99.6	99.6	97.6	97.0
LSTM	98.0	89.2	91.6	98.8	93.2	91.2	93.7	92.4
RF	67.6	61.2	62.8	57.2	84.4	52.4	64.3	63.7
SVM	36.0	36.0	32.0	40.0	32.0	32.0	34.7	34.4

Рисунок 4 – Сравнение точности

Достижение высокой точности для даже простого случайного леса возможно при предварительной чистке данных, так в одном из исследований [1]

(Рисунок 5) предположили, что разные поднесущие более или менее чувствительны к каким–то движениям, и обработав таким образом данные, то удается приблизиться или где–то даже обойти значения методов глубокого обучения:

Dataset	Source	Model	Metrics (%)				Time Cost (seconds)		
			Accuracy	Precision	Recall	F1-score	Total training	Total testing	Per sample
StanWiFi	(Yousefi et al. [56], 2017)	LSTM (2017)	90	-	-	-	-	-	-
	(Chen et al. [7], 2018)	ABLSTM (2018)	97.30	-	-	-	13,007.20	6.86	-
	(Yadav et al. [55], 2022)	CSITime (2022)	98.00	99.16	98.87	99.01	-	-	-
	(Salehinejad and Valaee et al. [41], 2022)	LiteHAR (2022)	93.00	-	-	-	157.80	5.46	0.013
	(Shalaby et al. [42] , 2022)	CNN-GRU (2022)	99.31	99.50	99.43	-	-	-	0.0033
	(Islam et al. [22], 2022)	STC-NLSTMNet (2022)	**99.88**	99.72	99.73	99.72	679	80	0.0028
	Proposed in this article	**AAE+RF**	99.84	**99.82**	**99.83**	**99.81**	**45.12**	**0.29**	**0.000086**
"MultiEnvironment" LOS (office)	(Alsaify et al. [4], 2021)	SVM (2021)	94.03	-	-	-	-	-	-
	(Alsaify et al. [2], 2022)	SVM (2022)	91.27	-	-	-	-	-	-
	(Islam et al. [22], 2022)	STC-NLSTMNet (2022)	**98.20**	**98.10**	**98.08**	**98.09**	710	87	0.0030
	Proposed in this article	**AAE+RF**	97.65	96.42	96.41	94.40	**49.20**	**0.35**	**0.000092**
"MultiEnvironment" NLOS	(Islam et al. [22], 2022)	STC-NLSTMNet (2022)	**94.68**	**94.57**	**94.55**	**94.56**	710	87	0.0030
	Proposed in this article	**AAE+RF**	93.33	93.12	93.07	93.14	**49.20**	**0.35**	**0.000092**

Рисунок 5 – Сравнение точности

2. Экспериментальная часть

2.1 Сбор данных

Были проведены два эксперимента.

Эксперимента 1 осуществлялся на расстояние меньше полуметра от приемника с передатчиком в комнате с другими двигающимися объектами. Среда представляет собой кабинет приблизительно 6*10 м, где находились за компьютерами ещё несколько человек, помимо испытуемого. Фиксировались следующие положения руки между установленными приемником и передатчиком, расстояние между которыми было около 1 м:

Класс 1 – Рука отсутствует внутри пространства

Класс 2 – Рука внутри пространства

Класс 3 – Рука двигается в пространстве медленно

Класс 4 – Рука резко выходит из пространства

Класс 5 – Рука плавно помещается в пространство

Эксперимент 2 проводился в комнате приблизительно 2,5*3,5 м. Расстояние между передатчиком и приемником приблизительно 2 м. Помимо испытуемого в комнате присутствовал только наблюдатель за компьютером для записи эксперимента. Были записаны следующие активности:

Класс 1 – Стояние

Класс 2 – Лежание

Класс 3 – Сидение

Класс 4 – Ходьба

Класс 5 – Бег

Устанавливается приемник и передатчик, передаются данные на компьютер, которые с помощью Python записываются в SQLite (Рисунок 6), предварительно фазы отчищаются. Суммарно получается 9 пар антенн и 56 поднесущих. Данные хранятся в качестве файлов json и представляют с собой массивы чисел.

id	id_packet	id_measurement	num_sub	ffa	fsa	fta	sfa	ssa	sta	tfa	tsa	tta
					Click here to define a filter							
1	1	1	0	145,523194027619	161,895027718581	29	104,656581255074	27,2029410174709	68,8767014308903	33,0151480384384	29,732137494637	29,529646
2	1	2	1	165,38742394753	184,32851108822	34,7131099154196	116,077560277601	31,2409987036266	76,0591874792257	38,2753184180093	32,5729949498047	33,105890
3	1	3	2	188,175450045961	203,03940504247	43,4165866921848	131,575073627188	32,5576411921994	84,9058301885094	42,5440947723653	37,1079506305589	37,696153
4	1	4	3	201,121853611188	222,899080303172	47,2016948848238	143,136298680663	37,4833296279826	92,4175308044962	45,6946386351834	39,4588393138977	41,109609
5	1	5	4	212,190951739229	242,662316810831	49,2442890089805	150,625363070102	41,0487515035476	97,6729235765982	47,5394572960189	42,190046219458	41,773197
6	1	6	5	220,911747084667	252,47970215445	47,5394572960189	157,917700084569	44,9444101084885	103,237590053236	51,2445119012758	46,0977222864644	43,931765
7	1	7	6	223,787845961303	263,138746671789	51,8652099195598	160,312195418814	43,4165866921848	103,585713300628	50	46,0108682813094	44,40720
8	1	8	7	225,328648866495	260,909946150008	47,8539444560216	162,003086390352	43,6577599058861	104,63746938836	54,1294744108974	45,0111097397076	44,18144
9	1	9	8	223,215142855497	263,539370872741	45,5411901469428	159,050306507092	44,2718872423573	105,209315176937	53,3666562565053	47,0425339453563	45,011109
10	1	10	9	227,107903869504	260,537905111713	46,2709412050371	159,154013458662	41,2310562561766	103,812330674155	50,9901951359278	48,0936586256442	43,046486
11	1	11	10	226,152603345617	258,089131890516	51	156,320184237353	42,5440947723653	105,118980208143	50,9901951359278	49,254441424099	43,011626
12	1	12	11	222,326336721496	260,376650258813	47,0106370941726	153,5512943612	46,0977222864644	104,043260233424	47,6759897642409	47,6759897642409	43,011626
13	1	13	12	213,459598050779	252,685575369866	48,0416485978573	150,651916682132	46,8187996428785	106,400187969759	48,6621002423858	45,3982378512647	43,046486
14	1	14	13	214,105114371423	253,671046830339	45,0444225182208	150,332963783729	45,6946386351834	104,403065089106	47,6759897642409	45,3982378512647	42,047592
15	1	15	14	211,596313767513	255,425135803037	43,1856457633784	148,273396130257	45,6179789118282	102,800778207171	45,7055795281058	46,3896540189728	43,011626
16	1	16	15	210,16184239771	254,348186547496	48,0416485978573	145,124084837769	47,4130783645188	100,801785698469	44,7213595499958	44,2831796509691	41,012193
17	1	17	16	212,51117617669	252,77064703007	48,2597140480546	145,013792447477	50,0899191454728	98,2344135219425	46,3249392876019	45,0998891351187	41,109609
18	1	18	17	206,421898063166	242,744721878767	48,2597140480546	143,003496460751	42,0475920832573	97,7445650662992	46,5725240887801	44,0113621693308	39,31920

Рисунок 6 – Пример хранения амплитуды по антеннам для эксперимента 1

В другой таблице хранится время приема пакета, id и класс. Было решено использовать для пробного обучения – 5 классов.

2.2 Анализ данных

Рассмотрим на примере эксперимента 1 процесс работы с данными.

Подключаем google disk:

```
from google.colab import drive
drive.mount('/content/drive/')
```

Импортируем библиотеки нужные:

```
import json
import time
from matplotlib import pyplot
import math
import numpy as np
import pandas as pd
import numpy as np
import pywt
import tensorflow as tf
from keras.regularizers import l1_l2, l1, l2
from tensorflow import keras
from tensorflow.keras.layers import Dense, Flatten, Dropout,
concatenate, LSTM, Conv2D, Embedding, Reshape, Bidirectional
from keras.models import Sequential
```

```
from keras.utils import plot_model
from keras.models import Sequential
from keras.layers import Conv1D, BatchNormalization,
GlobalMaxPooling1D, Dense, Activation, GlobalMaxPooling2D
```

Для чтения данных использовался следующий класс:

```
class CSIDataSet(object):
    def __init__(self, filename, visibleExamples=0,
max_examples=700):
        self.filename = filename
        self.data = []
        with open(filename, "r") as read_file:
            self.data = json.load(read_file)[:max_examples]
        print("Examples from dataset" + filename)
        for i in range(visibleExamples):
          print(str(i) + ": " + str(self.data[i]))
        self.packetCount = len(self.data)
        print("All count: "+str(self.packetCount))
    def getSetWith(self, count):
      if count <= len(self.data):
         return self.data[:count]
      else:
         print("Count exceeds the number of elements in the
dataset")
         return None
```

Записываем массив данных:

```
file_paths1 =
[f"//content/drive/MyDrive/Diplom/amplitude/classes/c1/data_class_
1_ampl_{i}.json" for i in range(1, 10)]
data_sets1a = [CSIDataSet(file_path).getSetWith(size) for
file_path in file_paths1]
```

Так делаем для каждого из 5 классов. В текущем виде data_sets1a имеет вид: (9, 700, 56). Далее приводим к виду (700, 504) массив каждого класса, а потом объединяем в один датасет:

```
data_s=np.concatenate((d1,d2,d3,d4,d5))
```

```
DsX=data_s
```

Размерность равна (3500, 504). DsX – это массив с признаками по всем 5 классам с 9 антенн и 56 поднесущих, который понадобится для обучения модели. Также есть массив DsY, содержащий метки класса для обучения модели, который заполнен 0 и 1, его размерность равна (3500, 5), где 5 – это количество классов.

Так как данные зашумленные их следовало сначала фильтровать, для это использовался вейвлет–фильтр из библиотеки pywt.

Вейвлет–фильтр — это математическая функция, используемая для разделения данной функции или сигнала непрерывного времени на компоненты разного масштаба. Обычно каждому компоненту масштаба можно назначить частотный диапазон. Затем каждый масштабный компонент может быть изучен с разрешением, соответствующим его масштабу.

В начале исследования был использован фильтр Хаара, так как исходя из других источников [2,3] был определен как наиболее эффективный. Потом было решено посмотреть также другие рекомендуемые вейвлет–фильтр и сравнить (Таблица 1). Брались 3 фильтра с разным порядком:

1. Haar – самый простой вейвлет–фильтр.

2. Symlets (Symmerical Wavelets) имеет более гладкие коэффициенты по сравнению с Haar. Порядок 2 и 20 – самый низкий и самый высокий

3. Daubechies (Daubechies Wavelets)– это асимметричные вейвлет–фильтры с большим количеством коэффициентов. Порядок 1 и 38 – самый низкий и самый высокий.

Таблица 1 – Сравнение фильтров для LSTM, Эксперимент 1

Фильтр	Точность
Haar (Haar)	0,876
sym2 (Symlets)	0,896
sym20 (Symlets)	0,899
db1 (Daubechies)	0,901
db38 (Daubechies)	0,901

Видно, что фильтр Haar имеет самую низкую точность в сравнение с другими. Было решено затем использовать db38, так как фильтры с более высоким порядком имеют более гладкие коэффициенты и лучше подходят для обработки более сложных сигналов.

```
DsX  = np.fft.fft(DsX)
wavelet = 'db38'
coeffs = pywt.wavedec(DsX, wavelet)
```

Создадим рисунки (Рисунок 7, Рисунок 8) для наглядного сравнения:

```
import matplotlib.pyplot as plt
d1f = (d1f - np.min(d1f)) / (np.max(d1f) - np.min(d1f))
plt.figure(figsize=(12, 6))
plt.plot(d1f)
plt.xlabel('Количество пакетов')
plt.ylabel('Фаза')
plt.title('График нормализованной фазы CSI сигнала')
plt.grid(True)
plt.show()
```

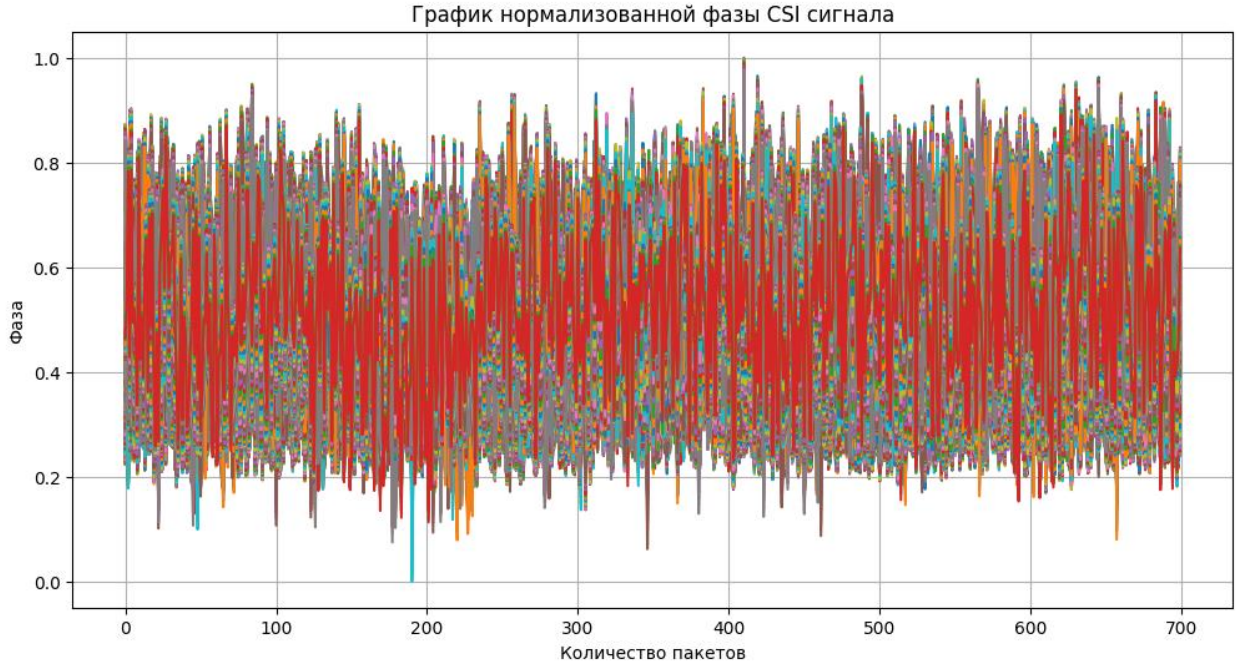

Рисунок 7 –Фазы, класс 1, без применения фильтра, эксперимент 1

Для фильтрации добавляется ещё пара строк кода:

```
d1f = np.fft.fft(d1f)
coeffs = pywt.wavedec(d1f, 'db38')
```

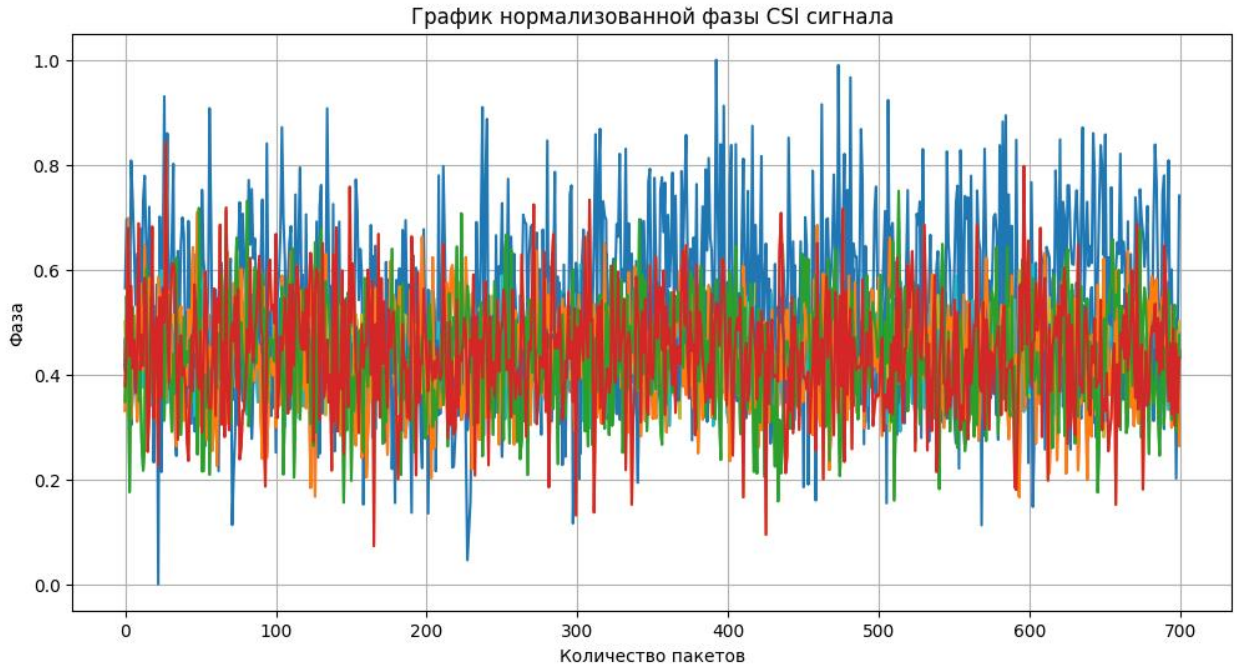

Рисунок 8 – Фазы, класс 1, с применением фильтра db38, эксперимент 1

2.3 Обучение

Далее используя библиотеку Keras создаем модель глубокого обучения. Были выбраны 5 моделей, которые будут обучаться, основывая на научно–исследовательских статьях, описываемых в пункте 1.2. Будет создан лист из 10 объектов класса модели `newModel`, чтоб в дальнейшем получить и сравнить результаты обучения для каждой:

```
model_count = 10
Models = list()
for i in range(model_count):
  model = newModel(input_shape=(None,504))
  Models.append(model)
```

Соответственно форма DsX должна быть приведена к (3500, 1, 504):

```
DsX = tf.expand_dims(DsX, axis=1)
```

Для определенной модели входной параметр формы может меняться как `input_shape=(None,1,504)`, в таком случае и операция для изменения формы будет проведена дважды чтоб иметь вид (3500, 1, 1, 504).

```
train_loss, val_loss, accuracy = list(), list(), list()
index = 0
st=time.time()
for model in Models:
  index += 1
  start_time = time.time()
  history = model.fit(DsX, DsY, epochs=100, batch_size=100,
validation_split=0.1, verbose=0)
  train_loss.append(history.history['loss'])
  val_loss.append(history.history['val_loss'])
  accuracy.append(history.history['accuracy'])
  print("--- %s-я эпоха пройдена ---" % (index))
  print("--- %s секунд ---" % (time.time() - start_time))
print("--- суммарно %s секунд ---" % (time.time() - st))
```

В этой части кода происходит процесс обучения моделей: на вход передается датасеты с признаками DsX и с метками класса DsY, который был до

этого заполнен 0 и 1, также устанавливаются значения для эпохи, батча и валидационной выборки.

Эпоха (epoch) – один полный проход обучающей выборки через модель глубокого обучения.

Batch size – количество примеров, которые будут одновременно подаваться на вход модели во время одной итерации обучения.

Validation_split – доля данных из обучающей выборки, которая будет использована для валидации модели во время обучения. В данном случае 10%.

Далее выводим точность для каждого объекта класса, получаем минимальную, среднюю и максимальную точность из списка объектов:

```
true_list = list()
tp, index = 0, 0
for model in Models:
  index += 1
  pred_DsY = model.predict(DsX)
  for i in range(len(pred_DsY)):
    tp = (tp + 1) if np.argmax(pred_DsY[i]) == np.argmax(DsY[i])
else tp
  true_list.append(tp/len(DsY))
  print(tp/len(DsY))
  tp = 0
print("true_list")
print(true_list)
print("Min:", '%.4f' % min(true_list))
print("Ave:", '%.4f' % (sum(true_list)/len(true_list)))
print("Max:", '%.4f' % max(true_list))
```

Рассмотрим каждую выбранную модель обучения по–отдельности.

2.3.1 LSTM

LSTM (Long Short–Term Memory): LSTM являются разновидностью рекуррентных нейронных сетей (RNN) и могут эффективно моделировать

временную зависимость в данных CSI. LSTM способны запоминать долгосрочные зависимости в последовательностях.

```
def newModel(input_shape):
   model = Sequential()
   model.add(BatchNormalization())
   model.add(GlobalMaxPooling1D())
   model.add(Reshape((1, 504)))
   model.add(LSTM(units=64, return_sequences=False))
   model.add(Dropout(0.2))
   model.add(Dense(5, activation='softmax'))
   model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
   return model
```

BatchNormalization(): Нормализация пакета данных помогает ускорить обучение модели и делает ее более устойчивой к различным инициализациям весов. Это особенно важно для сложных моделей, таких как LSTM, так как нормализация предотвращает возникновение проблем с градиентами.

GlobalMaxPooling1D(): Этот слой выполняет глобальное максимальное пулирование по временному измерению. Это позволяет модели обобщить наиболее важные признаки, извлеченные LSTM–слоем, в фиксированный размер вектора признаков.

Reshape((1, 504)): Этот слой преобразует выходной тензор из GlobalMaxPooling1D() в двумерную форму, которая соответствует входному размеру для LSTM–слоя.

LSTM(units=64, return_sequences=False): LSTM–слой с 64 скрытыми единицами. Установка `return_sequences=False` означает, что LSTM будет возвращать только последнее скрытое состояние, а не всю последовательность скрытых состояний.

Dropout(0.2): Слой Dropout случайно отключает 20% нейронов на этом уровне во время обучения, что помогает предотвратить переобучение модели.

Dense(5, activation='softmax'): Полносвязный выходной слой с 5 нейронами (так как 5 классов активности) и функцией активации softmax для получения вероятностей принадлежности к каждому классу.

Было проведено сравнение точности в зависимости от количества нейронов в слое LSTM:

Таблица 2 – Амплитуды, LSTM, haar, Эксперимент 1

units	BatchNormalization	GlobalMaxPooling1D	Время с	Точность
64	Да	да	388,5	0,904
200	Да	да	733,7	0,904

Точность до тысячных совпадала, было решено оставить 64 нейрона в угоду скорости обучения (Рисунок 9).

```
fig, (ax1, ax2) = pyplot.subplots(2, figsize=(12, 6), sharex=True)
loss_list = (train_loss)
ax1.plot(true_list)
ax1.set_ylabel("Точность (Accuracy)")
ax2.plot(loss_list)
ax2.set_ylabel("Ошибка (loss)")
ax2.set_xlabel("Модели")
```

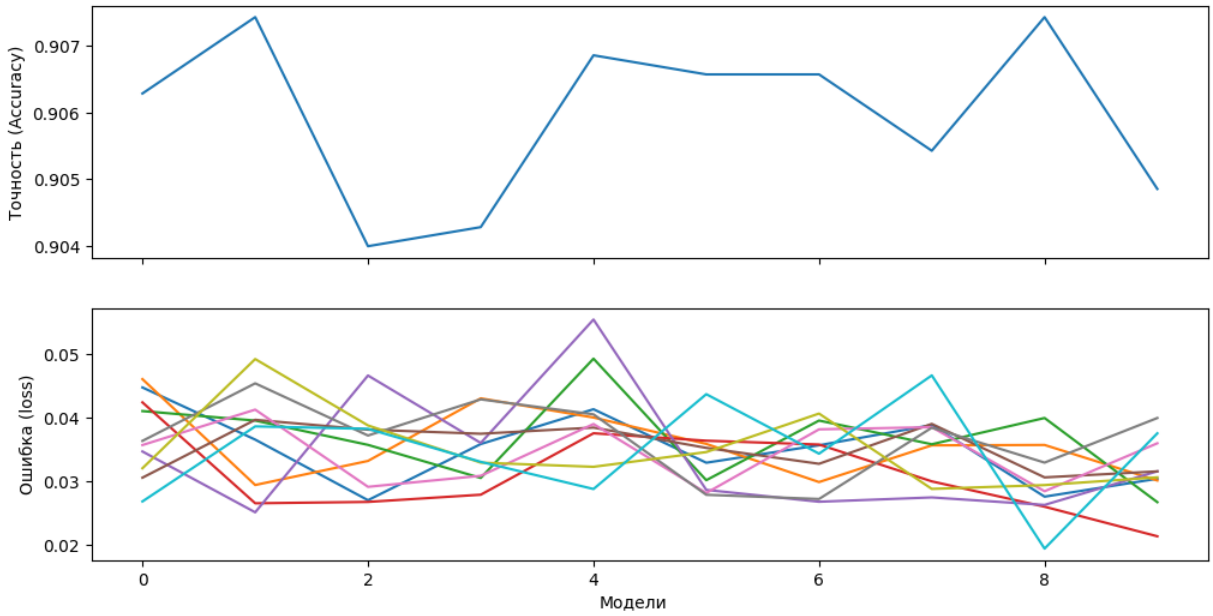

Рисунок 9 – пример построенного графика ошибок и точности для LSTM, Амплитуда, db38, Эксперимент 1

2.3.2 1D–CNN

1D–CNN (One–Dimensional Convolutional Neural Network): Эта модель хорошо подходит для обработки одномерных временных рядов, что подходит для данной задачи. 1D–CNN могут эффективно извлекать локальные признаки и улавливать временную зависимость в данных.

Вид модели был позаимствован из другой статьи [6] (Рисунок 10).

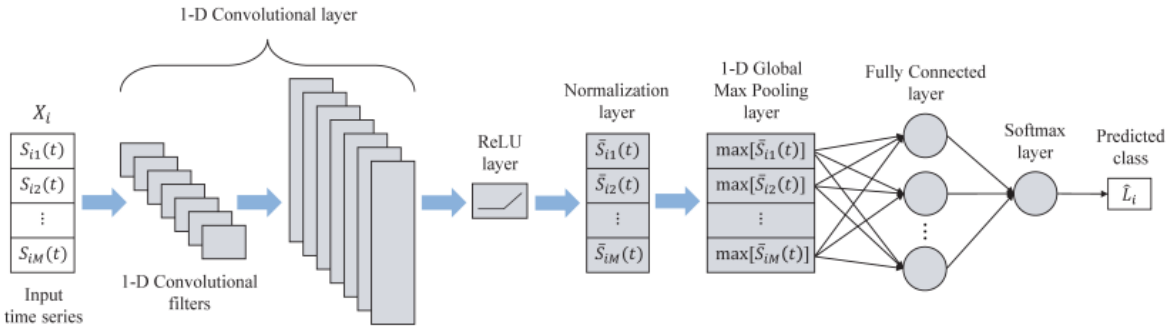

Рисунок 10 – Архитектура 1D–CNN

```
def newModel(input_shape):
    model = Sequential()
```

```
    model.add(Conv1D(filters=504, kernel_size=3, padding='same',
input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(BatchNormalization())
    model.add(GlobalMaxPooling1D())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(5, activation='softmax'))
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Были проведены сравнения по точности в зависимости от указывания размера ядра и пример

Таблица 3 – Амплитуды, 1D–CNN, haar, Эксперимент 1

Фильтр	Ядро	Время с	Точность
да	3	799,2	0,88
нет	3	737,7	0,65
да	1	378,1	0,87
да	5	1354,4	0,87

Разница между 1, 3 и 5 ядрами не значительна. В рамках исследования продолжалось использовать модель с 1 ядром для сокращения скорости обучения. Также проводится сравнение для демонстрации по точности для датасета, на котором был применен фильтр, и на котором не был.

2.3.3 1D–LSTM

1D–LSTM аналогична 1D–CNN за исключением добавления ещё одного слоя LSTM. Также взято из того источника [6] (Рисунок 11).

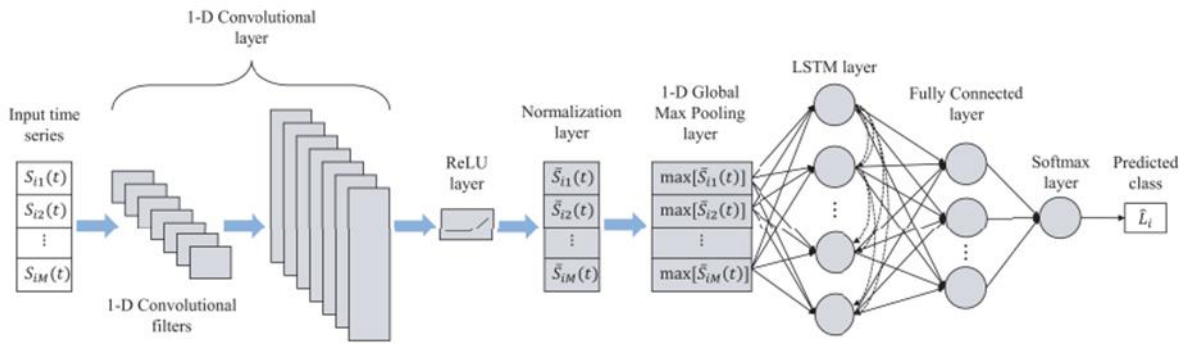

Рисунок 11 – Архитектура 1D–LSTM

```
def newModel(input_shape):
    model = Sequential()
    model.add(Conv1D(filters=504, kernel_size=1, padding='same',
input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(BatchNormalization())
    model.add(GlobalMaxPooling1D())
    model.add(Reshape((1, 504)))
    model.add(LSTM(units=64, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(5, activation='softmax'))
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Таблица 4 – Амплитуды, 1D–LSTM, haar, Эксперимент 1

Ядро	BatchNormalization	GlobalMaxPooling1D	Время с	Точность
1	да	да	805,1	0,899
1	нет	нет	760,6	0,20
3	да	да	981,3	0,892

Аналогично, существенной разницы между 1 и 3 ядром нет. Также демонстрируется как нормализация пакета с глобальным максимальным пулированием значит способствует точности модели.

2.3.4 2D–CNN

2D–CNN (Two–Dimensional Convolutional Neural Network) схож с 1D–CNN, основное различие между архитектурами 1D–CNN и 2D–CNN заключается в размерности обрабатываемых данных. 1D–CNN используется для последовательных данных, в то время как 2D–CNN используется для изображений и видеоданных. Поэтому в данном случае с помощью предварительной обработки данных мы подаем также CSI, но уже в виде изображения.

```
def newModel(input_shape):
    model = Sequential()
    model.add(Conv2D(filters=504, kernel_size=1, padding='same',
input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(BatchNormalization())
    model.add(GlobalMaxPooling2D())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(5, activation='softmax'))
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

2.3.5 BLSTM

Bi–LSTM (Bidirectional LSTM): Bi–LSTM – это расширение LSTM, которое использует две LSTM–сети, работающие в противоположных направлениях. Это позволяет модели учитывать как прошлые, так и будущие контексты, что может быть полезно для распознавания человеческой активности.

```
def newModel(input_shape):
```

```
    model = Sequential()
    model.add(BatchNormalization())
    model.add(GlobalMaxPooling1D())
    model.add(Reshape((1, 504)))
    model.add(Bidirectional(LSTM(units=64,
return_sequences=False)))
    model.add(Dropout(0.2))
    model.add(Dense(5, activation='softmax'))
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Таблица 5 – Амплитуды, BLSTM, haar, Эксперимент 1

units	BatchNormalization	GlobalMaxPooling1D	Время с	Точность
64	да	да	433,2	0,905
32	да	да	391,9	0,904

Для BLSTM было решено увеличить количество нейронов до 64, сравнив точность.

3. Сравнение результатов

Были созданы обобщающие таблицы по каждой архитектуре модели:

Таблица 6 – Амплитуды, фильтр db38, Эксперимент 1

Модель	Класс 1	Класс 2	Класс 3	Класс 4	Класс 5	Средняя	Время
LSTM	1	1	1	1	0,53	0,906	354,2
1D–CNN	0,998	0,998	1	0,998	0,52	0,905	714,0
1D–LSTM	0,992	0,992	0,997	0,991	0,517	0,892	428,8
2D–CNN	0,982	0,968	0,981	0,984	0,507	0,888	2718,02
BLSTM	1	1	1	1	0,52	0,905	464,7

Наилучшим образом для амплитуд себя показали архитектуры LSTM, BLSM, 1D–CNN, имеющие процент точности 90%, где 4 из 5 классов распознаются на 99% с погрешность в 0,002.

Таблица 7 – Фазы, фильтр db38, Эксперимент 1

Модель	Класс 1	Класс 2	Класс 3	Класс 4	Класс 5	Средняя	Время
LSTM	0,997	0,994	0,982	0,998	0,542	0,9004	726,66
1D–CNN	1	0,998	1	1	0,54	0,9067	800,68
1D–LSTM	1	0,994	0,994	0,998	0,541	0,9067	689,5
2D–CNN	1	1	1	1	0,535	0,9080	397,91
BLSTM	0,995	0,991	0,994	0,994	0,51	0,8992	713,13

Для фаз наилучшие результаты у архитектур 1D–CNN, 1D–LSTM и 2D–CNN, в то время как LSTM и BLSTM показали результат значительно хуже.

В большинстве обучений использовался именно датасет с амплитудой, но фазы и сочетание фаз с амплитудой также рассматривалось.

Для того чтоб «сложить» данные с амплитуды и фазы – необходимо было данные сначала нормализовать, так как временной ряд для амплитуды и фазы был в разном числовом диапазоне.

```
X_amplitude_norm = (DsXa - np.min(DsXa)) / (np.max(DsXa) -
np.min(DsXa))
X_phase_norm = (DsXf - np.min(DsXf)) / (np.max(DsXf) -
np.min(DsXf))
```

Таблица 8 – LSTM, фильтр db38, Эксперимент 1

Датасеты	Точность
Амплитуда	0,9047
Фаза	0,9047
Амплитуда (нормализованные)	0,9054
Фаза (нормализованные)	0,8997
Амплитуда+Фаза (нормализованные)	0,9010

Обучение датасета на амплитуде в среднем показало выше точность для архитектуры LSTM. Совмещение же двух датасетов не дало таких же высоких результатов.

Таблица 9 – Амплитуды, фильтр db38, Эксперимент 2

Модель	Стоять	Лежать	Сидеть	Идти	Бежать	Средняя	Время
LSTM	1	0,9996	1	0,9992	0,601	0,919	800,7
1D–CNN	0,991	0,995	0,978	0,976	0,587	0,903	2387,0
1D–LSTM	0,943	0,993	0,982	0,963	0,576	0,903	1452,8
2D–CNN	0,98	0,98	0,991	0,983	0,578	0,902	1192,5
BLSTM	1	1	1	1	0,614	0,9222	1390,57

BLSTM и LSTM показали наибольшую точность, как можно видеть по таблице – почти 92 % точности для обеих архитектур. Также 4 из 5 классов распознаются на все 100%, на следующем рисунке (Рисунок 12) это хорошо видно:

```
from sklearn.metrics import confusion_matrix,
classification_report
pred_DsY = Models[0].predict(DsX)
```

```
cm = confusion_matrix(np.argmax(DsY, axis=1), np.argmax(pred_DsY,
axis=1))
pyplot.imshow(cm, interpolation='nearest', cmap=pyplot.cm.Blues)
pyplot.title('Матрица ошибок')
pyplot.colorbar()
classes = ['Стоять', 'Лежать', 'Сидеть', 'Идти', 'Бежать']
tick_marks = np.arange(len(classes))
pyplot.xticks(tick_marks, classes, rotation=45)
pyplot.yticks(tick_marks, classes)
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
  for j in range(cm.shape[1]):
    pyplot.text(j, i, format(cm[i, j],
'd'),horizontalalignment="center",color="white" if cm[i, j] >
thresh else "black")
pyplot.tight_layout()
pyplot.show()
```

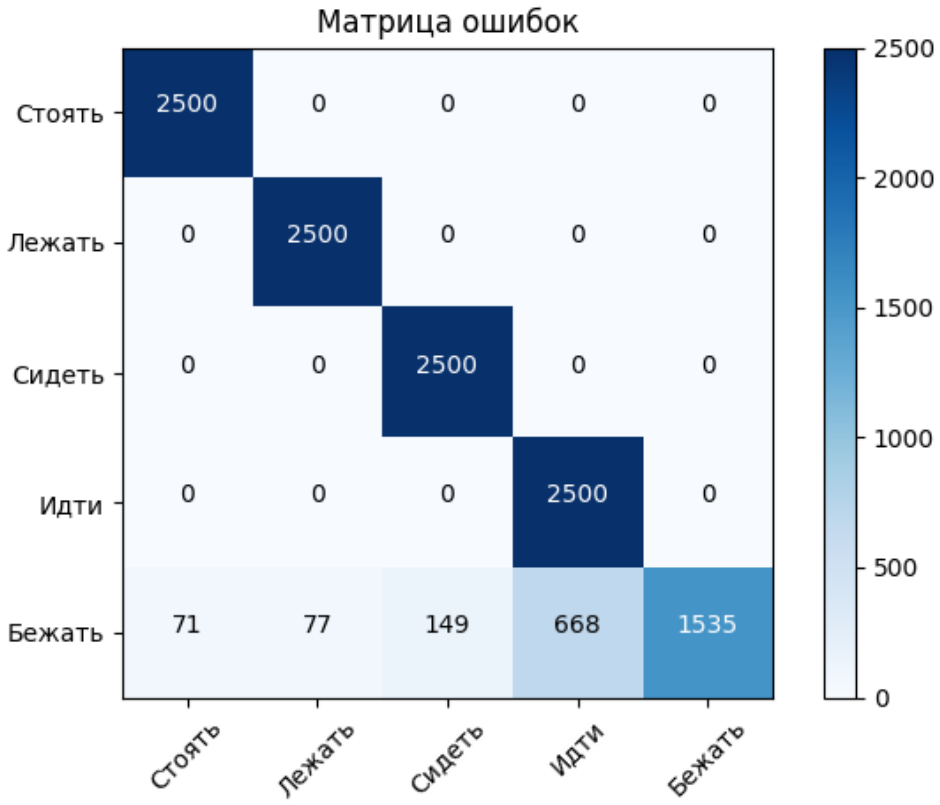

Рисунок 12 – Матрица ошибок для Амплитуда, BLSTM, db38, Эксперимент 2

Ходьба и бег осуществлялись на месте. Как можно увидеть по рисунку – бег часто принимается за ходьбу. Предполагаю, что как раз за счет того, что запись была на одном месте – класс 5 хуже распознавался, так как при обычном беге корпус имеет больший наклон и характер движения несколько иной, что и могло дать существенную разницу.

Таблица 10 – Фазы, фильтр db38, Эксперимент 2

Модель	Стоять	Лежать	Сидеть	Идти	Бежать	Средняя	Время
LSTM	0,7724	0,9024	0,902	0,866	0,592	0,8063	654,53
1D–CNN	0,8868	0,9696	0,8812	0,864	0,5732	0,9068	1907,39
1D–LSTM	0,992	0,996	0,9892	0,9964	0,6148	0,9171	1199,51
2D–CNN	0,9992	0,998	0,9988	1	0,6328	0,9213	745,44
BLSTM	0,9052	0,9448	0,9312	0,9408	0,616	0,8667	929,29

1D–LSTM и 2D–CNN показали лучший результат (Рисунок 13), в то время как LSTM и BLSTM, показавшие высокие значения для амплитуд, для фаз показали результат значительно хуже. Прослеживается тенденция, что для каждого типа данных точность для одних и тех же архитектур может значительно отличаться. В целом как и в эксперименте 1 вышло что LSTM и BLSTM имеют высокие показатели для амплитуд, в то время как для фаз подходят больше остальные: 2D–CNN, 1D–CNN, 1D–LSTM.

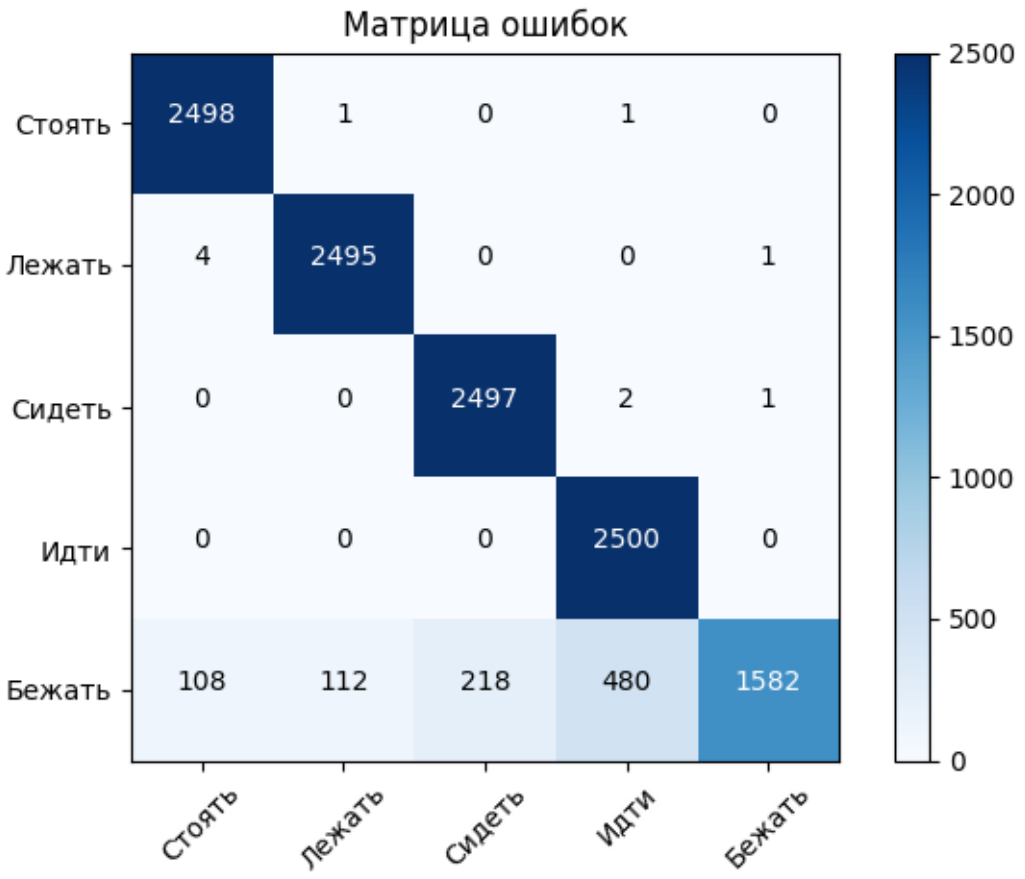

Рисунок 13 – Матрица ошибок для Фазы, 2D–CNN, db38, Эксперимент 2

Далее целью было попробовать ещё увеличить точность для отдельных архитектур и провести дополнительные сравнения.

Таблица 11 – Амплитуды, LSTM, фильтр db38, Эксперимент 1

Слои	Время с	Точность
LSTM(64), Dropout(0.2), Dense(504, activation='relu'), Reshape((1, 504)),LSTM(64), Dropout(0.2), Dense(200, activation='relu'), Dense(5, activation='softmax’)	848,20	0,9047
LSTM(64), Dropout(0.2), Dense(200,activation='relu'), Dense(5, activation='softmax’)	523,54	0,9048
LSTM(200),Dropout(0.2),Dense(200,activation='relu'), Dense(200,activation='relu'),Dense(5,activation='softmax’)	804,13	0,9053

Наибольший результат был достигнут с помощью последнего варианта архитектуры.

Также исследовалась эффективность добавления слоя регуляризации.

Слой Regularization (регуляризации) в нейронных сетях – это техника, используемая для предотвращения переобучения модели. Регуляризация добавляет дополнительный штраф к функции потерь, чтобы сделать модель более устойчивой и обобщающей.

Основные виды регуляризации:

1. L1–регуляризация (Lasso–регуляризация):
   - Добавляет к функции потерь сумму абсолютных значений весовых коэффициентов.
   - Приводит к разреженным (sparse) весам, то есть многие веса становятся равными нулю.
   - Эффективна для отбора признаков, когда нужно определить наиболее важные.
2. L2–регуляризация (Ridge–регуляризация):
   - Добавляет к функции потерь сумму квадратов весовых коэффициентов.
   - Приводит к уменьшению абсолютных значений весов, но не обязательно к их обнулению.
   - Эффективна для уменьшения переобучения, когда нужно сохранить все признаки.
3. L1–L2–регуляризация (Elastic Net):
   - Комбинация L1 и L2–регуляризации.
   - Позволяет сочетать преимущества L1 и L2–регуляризации.

Выбор коэффициента регуляризации λ является важным гиперпараметром, который нужно подбирать экспериментально. Слишком большое значение λ приведет к чрезмерной регуляризации и недообучению, а слишком маленькое –

к переобучению. Обычно λ подбирается с помощью кросс–валидации на отложенной части данных.

Ограничения на коэффициенты регуляризации:

– Они должны быть неотрицательными (λ >= 0).

– Для L1–L2 регуляризации обычно используют λ1 >= 0 и λ2 >= 0.

– Значения коэффициентов подбираются экспериментально, часто в логарифмическом масштабе (например, 0.0001, 0.001, 0.01, 0.1, 1.0).

Таблица 12 – Амплитуды, LSTM, фильтр db38, Эксперимент 2

Регуляризации	Время с	Точность
–	800,7	0,919
l1=0.01+l2=0.01	990,47	0,8626
l1=0.01	732,76	0,9067
l1=0.1	672,2	0,2421
l1=0.001	763,75	0,9198
l1=0,0001	699,03	0,9204
l1=0.00001	789,61	0,9200
l2=0.01	662,43	0,9216
l2=0.001	709,95	0,9206
l1=0.0001+l2=0.01	737,44	0,9203

Использование l2=0.01 или l1=0.0001+l2=0.01 для регуляризации LSTM показало себя наилучшим образом, однако изменение было незначительным.

Таблица 13 – Амплитуды, BLSTM, db38, Эксперимент 2

Регуляризации	Время с	Точность
–	1361,04	0,9221
l1=0.0001, l2=0.01	1166,57	0,9215
l1=0.0001	1314,97	0,9217
l2=0.01	1252,36	0,9211

Те же самые параметры для регуляризации не улучшили точность для BLSTM.

Сравнивались различные гиперпараметры для эксперимента 1, где изначально по умолчанию epochs=100 и batch_size=100:

Таблица 14 – Амплитуды, LSTM, db38, Эксперимент 1

Batch_size=100		
epochs	Время с	Точность
10	135,14	0,8243
50	376,15	0,9043
100	767,09	0,9056
200	1373,43	0,9050

Так для batch_size=100 выбранные изначально epochs =100 были наиболее оптимальны с точки зрения точности и времени обучения модели. Затем были изменены размеры batch_size при оставленных размерах epochs.

Таблица 15 – Амплитуды, LSTM, db38, Эксперимент 1

Epochs=100		
Batch_size	Время с	Точность
100	767,09	0,9056
250	718,36	0,9053
500	487,38	0,9056

Точность для batch_size=100 и batch_size=500 идентична при epochs=100, но время занимает меньше второй вариант.

Для эксперимента 2 значение batch_size изначально был выбран как 500, в отличие от эксперимента 1, где это значение равно 100. Объясняется это тем что размер больше в рамках одного класса: (2500, 504). Соответственно на вход передается датасет с размером: (12500, 1, 504). Для epochs сохранилось значение 100, но было решено сравнить точность при других значениях параметра.

Таблица 16 – Фазы, LSTM, db38, Эксперимент 2

Batch_size=500		
epochs	Время с	Точность
100	721,02	0,8998
150	811,69	0,9144
200	1288,28	0,9203

Результат значительно вырос при epochs=200, однако и время обучения сильно увеличилось. При сохранение данного значения epochs было решено сравнить точность при изменении Batch_size.

Таблица 17 – Фазы, LSTM, db38, Эксперимент 2

Epochs=200		
Batch_size	Время с	Точность
100	1336,20	0,8209
350	656,02	0,8759
500	721,02	0,8998
700	712,82	0,8988

Результат подтвердил, что использование batch_size=500 было наиболее оптимальным решением.

Таблица 18 – Амплитуды, LSTM, фильтр db38, Эксперимент 2

Слои	Время с	Точность
LSTM(200),Dropout(0.2),Dense(200, kernel_regularizer=l2(0.01), activation='relu'), Dense(200, activation='relu') ,Dense(5,activation='softmax’)	3012,19	0,9227
LSTM(200),Dropout(0.2),Dense(200, kernel_regularizer=l2(0.01), activation='relu'), Dense(5,activation='softmax’)	2641,48	0,9225

Точность незначительно увеличилась в первом варианте.

Теперь за счет полученных гиперпараметров была проведена попытка улучшить результаты:

Таблица 19 – фильтр db38, Эксперимент 2

Модель	Датасет	Время с	Точность
1D–CNN	Фазы	3783,94	0,9170
1D–LSTM	Фазы	2146,8	0,9234
2D–CNN	Фазы	1372,17	0,9245
LSTM	Амплитуды	3012,19	0,9227
BLSTM	Амплитуды	1361,04	0,9221

Значение точности для каждой из архитектур увеличилось в сотых. Однако эти изменения весомо увеличили скорость обучения в сравнение с Таблицей 9 и Таблицей 10.

Заключение

В настоящее время устройства Wi–Fi, поддерживающие стандарт беспроводной связи 802.11, широко распространены, и сигналы CSI могут предоставлять значимую информацию об окружающей обстановке, за счет своей относительной дешевизны и простоты установки, могут конкурировать с использованием фото–видео устройств для распознания человеческой активности или датчиками движения.

Изучаемые по теме статьи послужили материалом для выступления на научной конференции «Зворыкинские чтения» с темой «Сравнение использования алгоритмов работы с CSI для задачи HAR».

Были проведены два эксперимента, показавшие достаточно высокий результат. В рамках эксперимента сравнивались 5 архитектур для обучения нейронной сети: LSTM, 1D–CNN, 1D–LSTM, 2D–CNN, BLSTM. Были проведены сравнения точности за счет изменения гиперпараметров: batch_size, epochs, регуляризации. Для эксперимента 1 и 2 эффективные значения batch_size и epochs отличались, регуляризация также для LSTM и BLSTM имела разную ценность.

Также проводилось изменение структуры отдельных архитектур: размер ядра, количество нейронов, добавление отдельных слоев. В большинстве своем многие сравнения проводились на архитектуре LSTM при датасете амплитуд. Сравнение точности при использование разных вейвлет–фильтров выявил также что вместо используемого изначально фильтра haar, куда эффективнее использовать db38.

В среднем для датасета амплитуд наилучшим образом себя показали LSTM и BLSTM, имея приблизительно 90 % точности для эксперимента 1 и 92% для эксперимента 2. Для датасета фаз с аналогичными значениями вышли архитектуры 1D–CNN, 1D–LSTM и 2D–CNN. При этом точность для 4 из 5 классов достигала 96–100%. В случае эксперимента 2 есть подозрение, что класс

«бег» путался с классом «ходьба» за счет особенностей записи данной активности, а именно – осуществляя её на месте.

Объединение датасета фаз и амплитуд не принесло видимых улучшений, однако сравнение проводилось только для LSTM.

Все поставленные цели работы выполнены. В качестве результата работы были собраны таблицы с данными и рисунки.

Список используемой литературы

1. Efficient Wi–Fi–Based Human Activity Recognition Using Adaptive Antenna Elimination [Электронный ресурс]. 2023. URL: https://www.researchgate.net/publication/374244149_Efficient_Wi–Fi–Based_Human_Activity_Recognition_Using_Adaptive_Antenna_Elimination

2. Towards CSI–based diversity activity recognition via LSTM–CNN encoder–decoder neural network [Электронный ресурс]. 2020. URL: https://www.researchgate.net/publication/347579677_Towards_CSI–based_diversity_activity_recognition_via_LSTM–CNN_encoder–decoder_neural_network

3. Differential Channel State Information based Human Activity Recognition in IoT Networks [Электронный ресурс]. 2020. URL: https://www.researchgate.net/publication/341646847_Differential_Channel–State–Information–Based_Human_Activity_Recognition_in_IoT_Networks

4. LSTM–CNN network for human activity recognition using WiFi CSI data [Электронный ресурс]. 2021. URL: https://www.researchgate.net/publication/351145259_LSTM–CNN_network_for_human_activity_recognition_using_WiFi_CSI_data

5. CSI–HC: A WiFi–Based Indoor Complex Human Motion Recognition Method [Электронный ресурс]. 2020. URL: https://www.sci–hub.ru/10.1155/2020/3185416

6. A Fast Deep Learning Technique for Wi–Fi–Based Human Activity Recognition [Электронный ресурс]. 2022. URL: https://www.researchgate.net/publication/367607860_A_FAST_DEEP_LEARNING_TECHNIQUE_FOR_WI–FI–BASED_HUMAN_ACTIVITY_RECOGNITION

7. WiFi CSI based passive human activity recognition using attention based BiLSTM [Электронный ресурс]. 2018. URL: https://www.researchgate.net/publication/328621270_WiFi_CSI_Based_Passive_Human_Activity_Recognition_Using_Attention_Based_BLSTM

8. DensePose From WiFi [Электронный ресурс]. 2022. URL: https://arxiv.org/abs/2301.00250v1