МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гродненский государственный университет им. Янки Купалы»

Физико-технический факультет

|  |  |
| --- | --- |
| Кафедра информационных  систем и технологий | ДОПУЩЕН К ЗАЩИТЕ  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.  Зав. Кафедрой информационных  систем и технологий  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ю.Р.Бейтюк |

Специальность: I-38.02.01 Информационно-измерительная техника

КУРСОВОЙ ПРОЕКТ

Тема: Реализация классификатора для удаленной диагностики физического состояний человека на основе нейронной сети LSTM

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

|  |  |  |
| --- | --- | --- |
| Разработал: |  | Ступакевич М.Р. |
|  | (подпись, дата) |  |
| Руководитель проекта: |  | Ассанович Б.А. |
|  | (подпись, дата) |  |

2021

Содержание

[ВВЕДЕНИЕ 3](#_Toc72835440)

[1. Обзор методов классификации состояний человека 5](#_Toc72835441)

[1.1 Обзор исследований по определению падения человека 5](#_Toc72835442)

[1.2 Рекуррентная нейронная сеть LSTM 8](#_Toc72835443)

[1.3 Описание используемых данных 12](#_Toc72835444)

[2. Реализации системы классификатора 14](#_Toc72835445)

[2.1 Структурная схема системы 14](#_Toc72835446)

[2.2 Используемое программное обеспечение 14](#_Toc72835447)

[2.3 Блок-схема и описание работы алгоритма 17](#_Toc72835448)

[ЗАКЛЮЧЕНИЕ 26](#_Toc72835449)

[Используемые источники 27](#_Toc72835450)

[Приложение 28](#_Toc72835451)

# ВВЕДЕНИЕ

Падение определяется как событие, в результате которого человек оказывается непреднамеренно лежащим на земле, полу или каком-либо другом более низком уровне. Падения являются значительной проблемой общественного здравоохранения во всем мире. По оценкам, ежегодно происходит 646 000 смертельных падений, что делает падения второй по значимости причиной смерти от непреднамеренных травм после дорожно-транспортных травм. Ежегодно происходит 37,3 миллиона падений с достаточно серьезными последствиями, при которых требуется медицинская помощь. Актуальность проблемы заключается в том, что во всех регионах мира самые высокие показатели смерти отмечаются среди людей старше 60 лет. Самая высокая распространенность таких падений наблюдается среди людей в возрасте 65 лет и старше. Кроме того, люди, пережившие падения и страдающие от инвалидности, особенно пожилые люди, подвергаются значительному риску возникновения необходимости в последующем длительном уходе и помещении в специальные учреждения.

Регистрирование падения человека – задача, решение которой возможно при современных вычислительных мощностях компьютеров. В данном курсовом проекте будут рассмотрены методы диагностики состояний человека и будет предложена программная реализация определения падения на видео, что полезно в отслеживании состояний пожилых людей и людей, страдающих разного рода заболеваниями.

Целью данного курсового проекта является разработка модели нейронной сети, определяющей события «Упал / Не упал» на видео на основе размеченных данных, описывающих состояние человека последовательностями характеристик (Например, координат и длин сторон прямоугольника, описывающего силуэт человека, полученных из видео, снятого на обычную камеру, с помощью нейронной сети; координат ключевых точек частей тела человека; данных с акселерометра и др.).

Для разработки модели необходимо решить следующие задачи:

1. Изучить методы диагностики состояния человека, выбрать инструментарий для реализации;
2. Подготовить набор данных (датасет) отображающих соответствие параметров измеряемых физических пар-ров состоянию человека.
3. Выполнить предобработку и нормировку данных для обучения и нейронной сети как классификатора, разработать ПО для реализации
4. Разработать структуру нейронной сети типа LSTM для обучения и классификации состояний человека;
5. Произвести обучение модели классификатора. Получить и проанализировать результаты тестирования.

# 1. Обзор методов классификации состояний человека

## **1.1 Обзор исследований по определению падения человека**

В настоящее время есть несколько методов для определения состояний человека, использующих для этого данные с акселерометра, видеокамеры и устройств, определяющих глубину изображения, работающих на так называемом ToF-датчике или лидаре.

Далее приведены два популярных исследования по определению падения человека:

1. В работе «Обнаружение падения человека на встроенной платформе с использованием карт глубины и беспроводного акселерометра», написанной сотрудниками Междисциплинарного центра вычислительного моделирования Жешувского университета в Польше Богданом Кволеком и Михалом Кепским в рамках журнала «Компьютерные методы и программы в биомедицине» [1] описан алгоритм определения падения с помощью беспроводного акселерометра, прикрепленному к одежде человека в районе правого бедра и камеры Kinect. На веб-странице работы предложен датасет (набор данных), включающий в себя 30 записей падений с двух RGB камер, двух камер Kinect, акселерометра и файла, описывающего синхронизации камер и акселерометра (Рисунок - 1.1). И 40 записей различных других активностей человека.

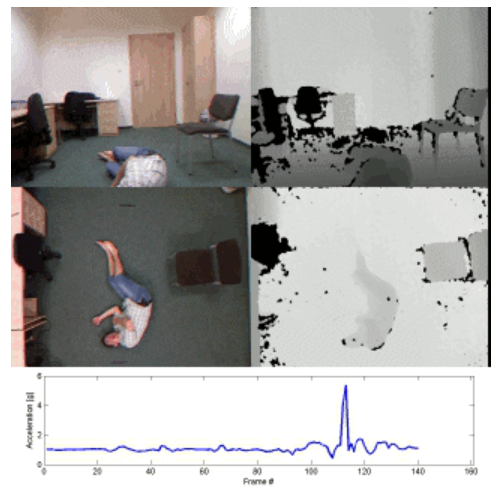


Рисунок – 1.1 Пример содержания датасета

Для классификации состояния человека, исходя из полученных данных, используется SVM-классификатор (Классификатор методом опорных векторов), являющийся линейным классификатором (Рисунок - 1.2). Метод опорных векторов – метод определения так называемых опорных векторов и построении двух параллельных «гиперплоскостей», разделяющих объекты в пространстве на классы. Алгоритм основан на допущении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

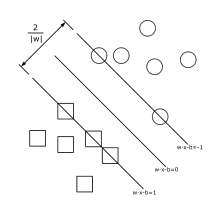


Рисунок - 1.2 Графическое представление разделения объектов на классы с помощью SVM-классификатора

2. Следующий проект представляет собой сотрудничество между Отделом компьютерного зрения (COMVIS) Международного научно-исследовательского института MICA (MICA) и Технологическим институтом почты и телекоммуникаций (PTIT) [2]. Целью этого проекта является создание непрерывного мультимодального многовидового набора данных о действиях человека при падении и разработка алгоритмов распознавания падения человека. Набор данных состоит как из обычных ежедневных действий, так и из моделируемых падений, полученного одновременно с камер Microsoft Kinect и беспроводных акселерометров. Набор данных CMDFALL содержит 20 действий, выполненных 50 людьми (30 мужчинами и 20 женщинами в возрасте 21-40 лет).

Схема мультимодальной системы сбора данных, используемой для сбора данных CMDF. Каждое действие одновременно фиксировалось двумя различными сенсорными системами: семью камерами Microsoft Kinect и двумя беспроводными акселерометрами (Рисунок – 1.3-1.4).

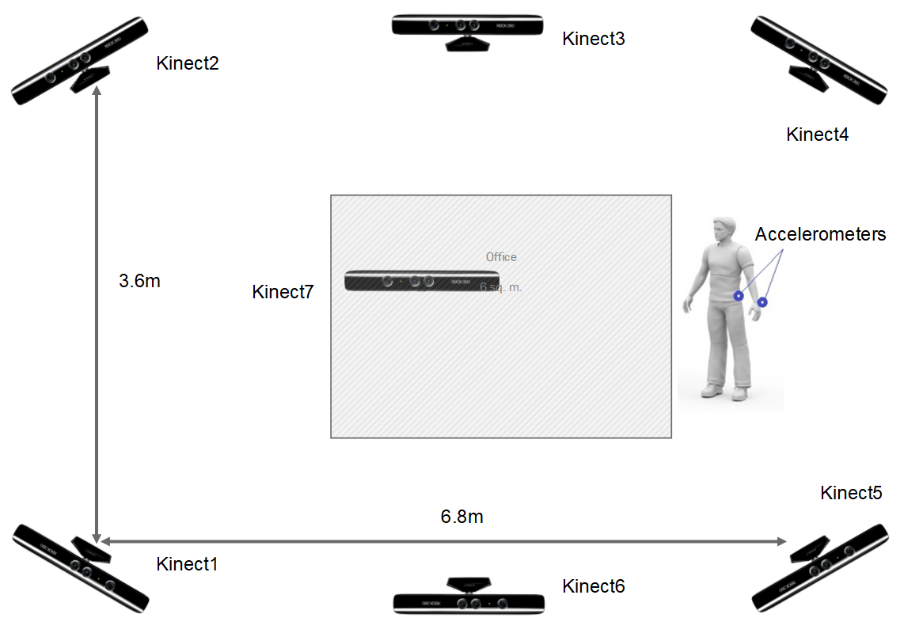


Рисунок – 1.3 Схема системы сбора данных

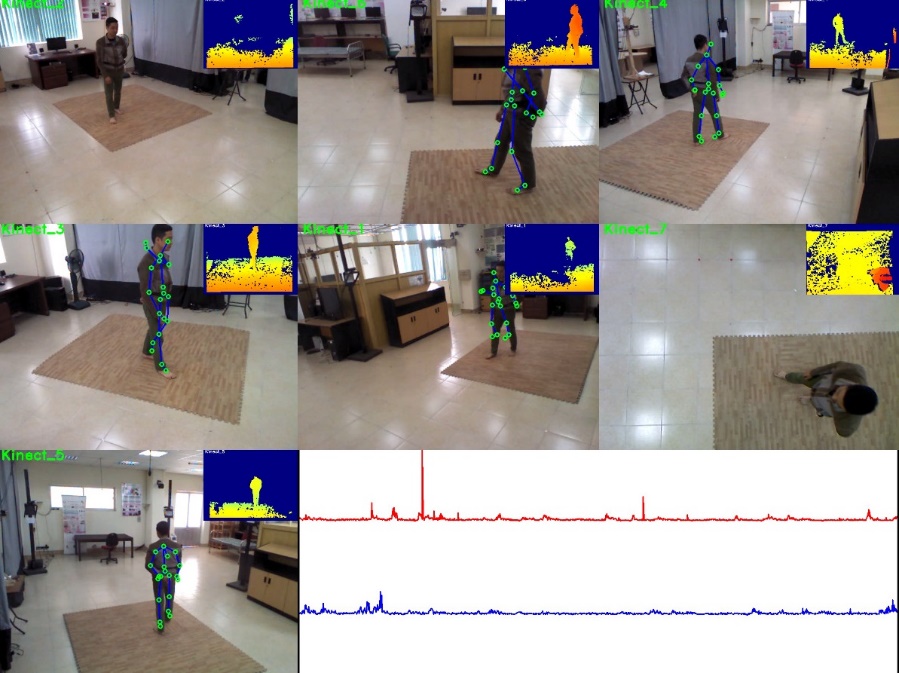


Рисунок – 1.4 Пример содержания датасета

## **1.2 Рекуррентная нейронная сеть LSTM**

Долгая краткосрочная память (Long short-term memory; LSTM) – особая разновидность архитектуры рекуррентных нейронных сетей, способная к обучению долговременным зависимостям. Они были представлены [Зеппом Хохрайтер и Юргеном Шмидхубером (Jürgen Schmidhuber) в 1997 году](http://deeplearning.cs.cmu.edu/pdfs/Hochreiter97_lstm.pdf), а затем усовершенствованы и популярно изложены в работах многих других исследователей. Они прекрасно решают целый ряд разнообразных задач и в настоящее время широко используются [3].

LSTM разработаны специально, чтобы избежать проблемы долговременной зависимости. Запоминание информации на долгие периоды времени – это их обычное поведение, а не что-то, чему они с трудом пытаются обучиться.  
На рисунке – 1.5 изображена структурная схема LSTM слоя.

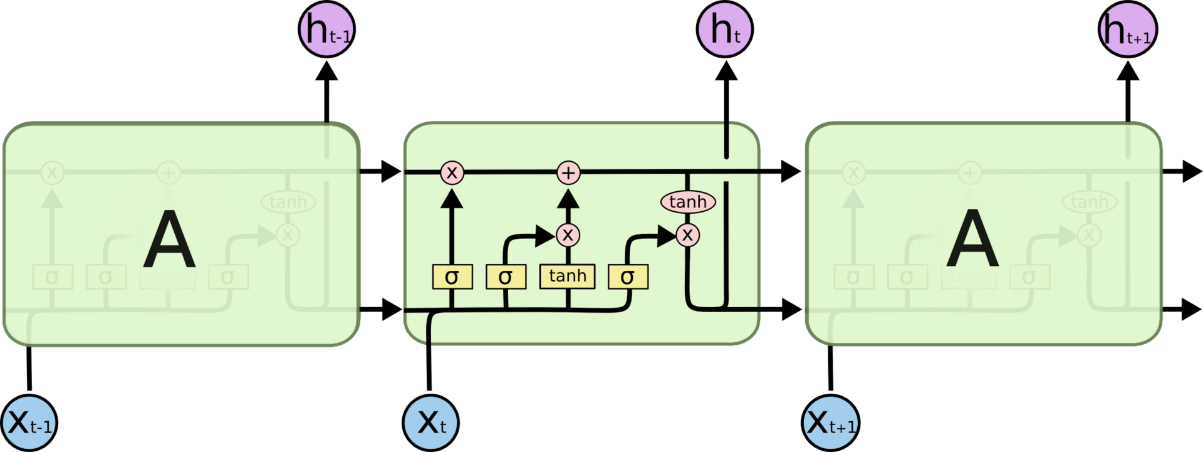


Рисунок – 1.5 Структурная схема LSTM слоя

Ключевой компонент LSTM – это состояние ячейки (cell state) – горизонтальная линия, проходящая по верхней части схемы (Рисунок - 1.6).  
  
 Состояние ячейки напоминает конвейерную ленту. Она проходит напрямую через всю цепочку, участвуя лишь в нескольких линейных преобразованиях. Информация может легко течь по ней, не подвергаясь изменениям.

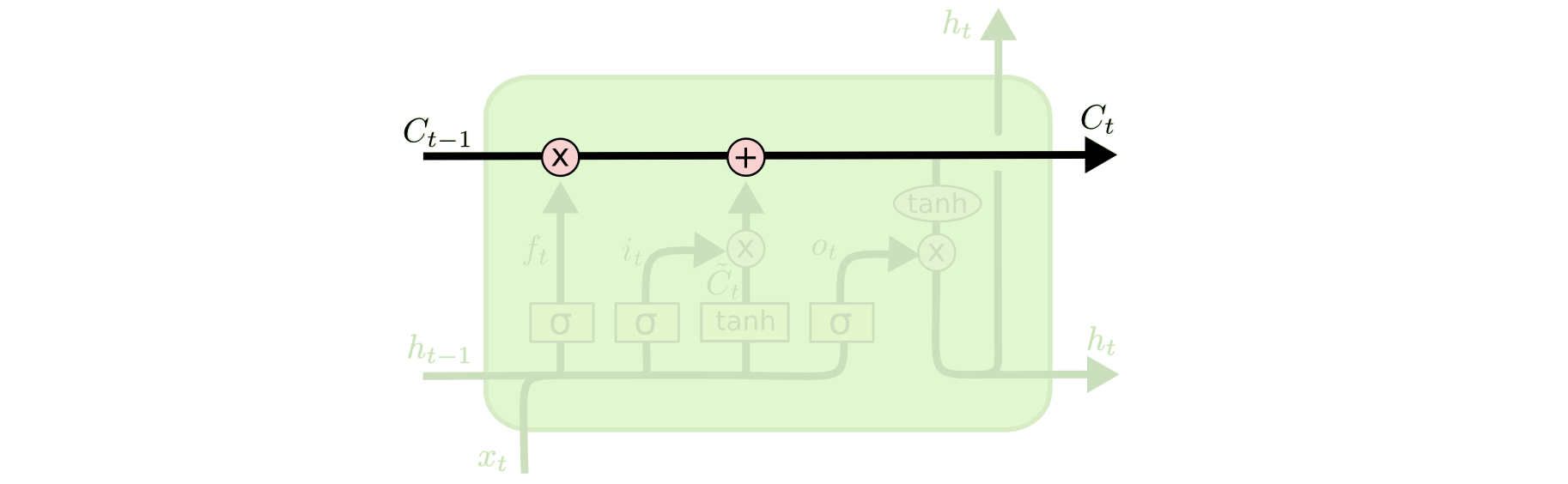


Рисунок - 1.6 Состояние ячейки

Тем не менее, LSTM может удалять информацию из состояния ячейки; этот процесс регулируется структурами, называемыми фильтрами (gates).  
 Фильтры позволяют пропускать информацию на основании некоторых условий. Они состоят из слоя сигмоидальной нейронной сети и операции поточечного умножения (Рисукок – 1.7 Сигмоидальный слой).

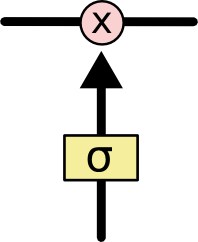


Рисунок - 1.7 Сигмоидальный слой

Сигмоидальный слой возвращает числа от нуля до единицы, которые обозначают, какую долю каждого блока информации следует пропустить дальше по сети. Ноль в данном случае означает “не пропускать ничего”, единица – “пропустить все”.   
  
 В LSTM три таких фильтра, позволяющих защищать и контролировать состояние ячейки.   
 Первый шаг в LSTM – определить, какую информацию можно выбросить из состояния ячейки. Это решение принимает сигмоидальный слой, называемый “слоем фильтра забывания” (forget gate layer) (Рисунок – 1.8). Он смотрит на ht−1 и xt и возвращает число от 0 до 1 для каждого числа из состояния ячейки Ct−1. 1 означает “полностью сохранить”, а 0 – “полностью выбросить”.

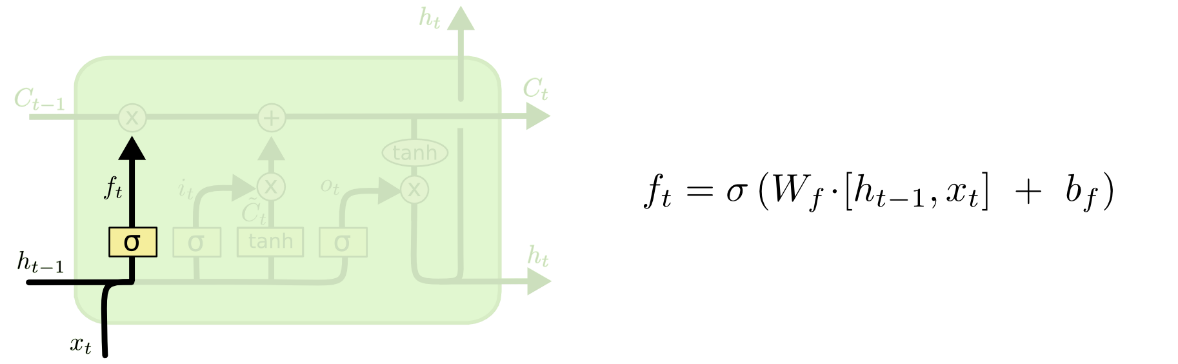


Рисунок – 1.8 Слой фильтра забывания

Следующий шаг – решить, какая новая информация будет храниться в состоянии ячейки. Этот этап состоит из двух частей. Сначала сигмоидальный слой под названием “слой входного фильтра” (input layer gate) определяет, какие значения следует обновить (Рисунок – 1.9). Затем tanh-слой строит вектор новых значений-кандидатов Ct, которые можно добавить в состояние ячейки.

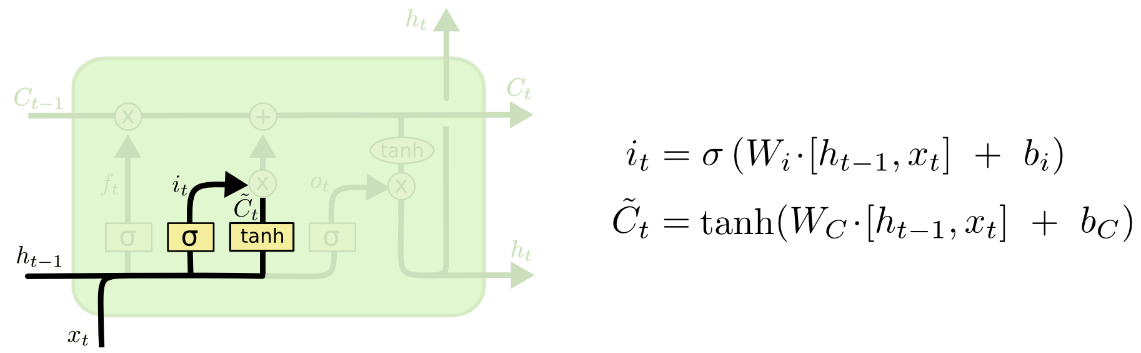


Рисунок – 1.9 Слой входного фильтра

Настало время заменить старое состояние ячейки Ct−1на новое состояние Ct.   
 Необходимо умножить старое состояние на ft, забывая то, что решено забыть. Затем прибавляется it∗Ct. Это новые значения-кандидаты, умноженные на t – на сколько следует обновить каждое из значений состояния (Рисунок - 1.10).

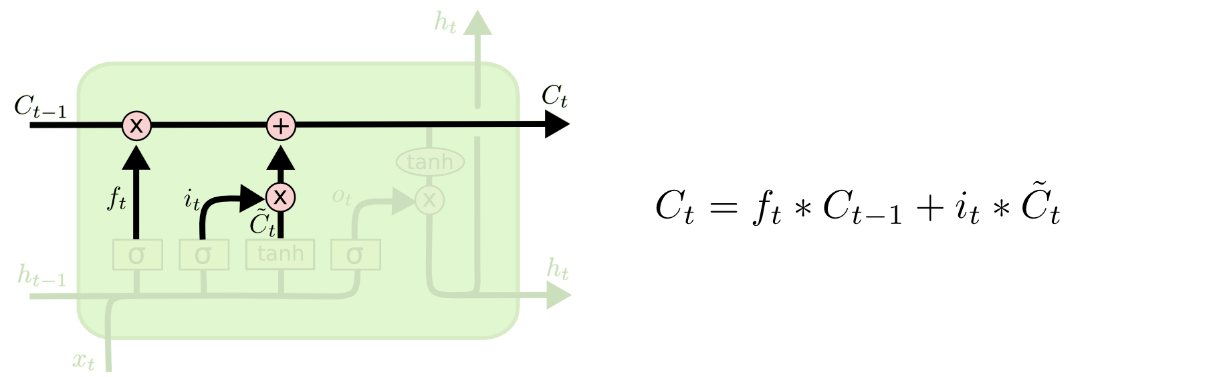


Рисунок – 1.10 Слой обновления состояния

Наконец, нужно решить, какую информацию необходимо получать на выходе. Выходные данные будут основаны на состоянии ячейки, к ним будут применены некоторые фильтры. Сначала применяется сигмоидальный слой, который решает, какую информацию из состояния ячейки будет нужно выводить. Затем значения состояния ячейки проходят через tanh-слой (слой с функцией активации «Гиперболический тангенс»), чтобы получить на выходе значения из диапазона от -1 до 1, и перемножаются с выходными значениями сигмоидального слоя, что позволяет выводить только требуемую информацию.

Именно поэтому в решении поставленной задачи следует использовать сеть LSTM, выделяя, запоминая и используя только необходимые данные из последовательности при обучении и последующем использовании модели для предсказания результата.

## **1.3 Описание используемых данных**

Чтобы нейронная сеть смогла обнаружить падения, необходим набор данных в реалистичной настройке видеозащиты с помощью одной камеры. В сети интернет был найден датасет, собранный в Бургундском университете во Франции [4]. Датасет включает в себя видео и аннотации к ним. Частота кадров видео составляет 25 кадров в секунду, а разрешение - 320×240 пикселей. Видеоданные иллюстрируют основные трудности реалистичных видеопоследовательностей, которые мы можем найти в домашней обстановке для пожилых людей, а также в простейшей офисной комнате. Видеопоследовательности содержат переменную освещенность и типичные трудности, такие как окклюзии или загроможденный и текстурированный фон. Актеры выполняли различные обычные ежедневные действия и падения. Набор данных содержит 191 видео, которые были аннотировали для целей оценки с дополнительной информацией, представляющей истинность положения падения в последовательности изображений. Затем каждый кадр каждого видео аннотируется: локализация тела определяется вручную с помощью ограничительных рамок (Рисунок - 1.11). Эта аннотация позволяет оценить классификационные признаки независимо от автоматического обнаружения тела.

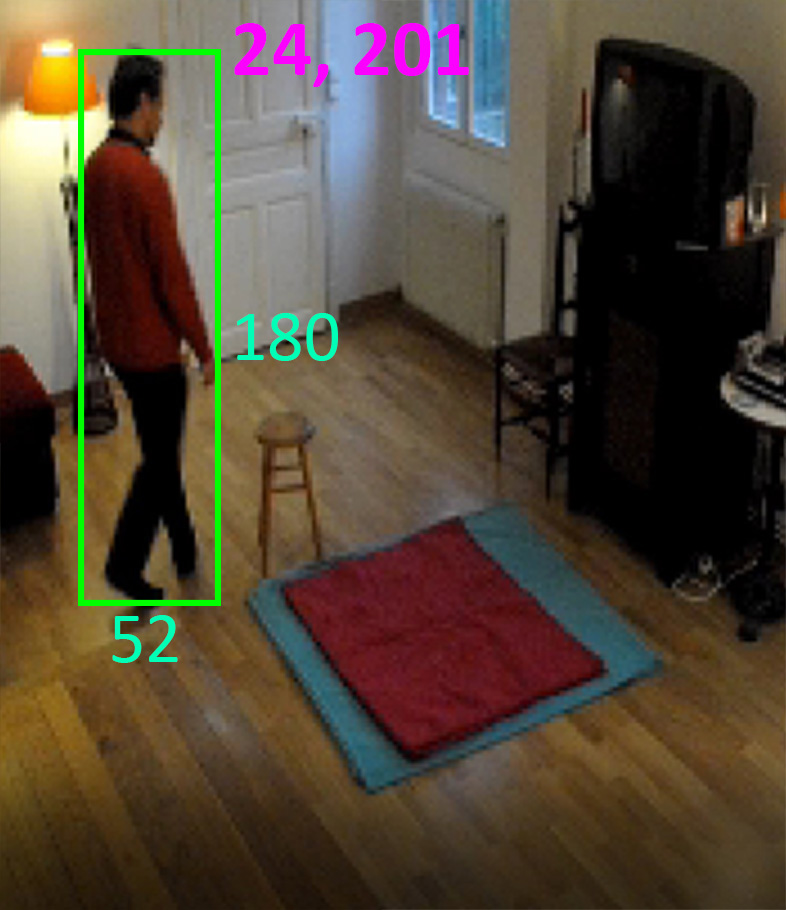


Рисунок - 1.11 Ограничительная рамка (Координаты левого нижнего угла - розовый, длины сторон в пикселях - голубой).

Как правило, несколько доступных наборов данных, предназначенных для обнаружения падений, используют одно и то же место для тестирования и обучения. Таким образом, это не позволяет оценить устойчивость метода к изменению местоположения между тренировочными данными и данными для тестирования. Чтобы оценить эту надежность, в датасете содержатся видео из разных мест, что позволило определить несколько протоколов оценки («Дом», «Кофейня», «Офис» и «Лекционный зал»).

# 2. Реализации системы классификатора

## **2.1 Структурная схема системы**

Система, выполняющая поставленную цель должна на входе получить данные и вывести предсказание «Упал / Не упал» на выходе. Для этого в системе должны быть реализованы следующие функции:

D:\загрузки\Untitled Diagram (1).jpg

Рисунок - 2.1 Структурная схема системы

## **2.2 Используемое программное обеспечение**

Решение разрабатывается на языке Python и дополнительных библиотек для работы с данными и моделирования нейронной сети с использование нескольких интерпретаторов: стандартного Python и среды PyCharm от JetBrains для получения данных и IPython в локальной веб-среде Jypyter для обработки данных, различного рода преобразований, моделирования структуры рекуррентной нейронной сети, проведения экспериментов и анализа результатов, т.к. IPython позволяет выполнять программный код последовательно по частям и визуально отслеживать результаты на всех этапах выполнения программы.

Используемые Python библиотеки:

OS ­­­­­­­— стандартная библиотека Python, позволяющая работать с операционной системой. В программе использовался метод listdir, возвращающий список файлов в директории. Это необходимо было для чтения файлов аннотаций и извлечения необходимых данных.

Pickle — модуль, реализующий мощный алгоритм сериализации и десериализации объектов Python. В данном модуле использовались методы dump и load для преобразования объектов Python в байты и сохранения в файлы. Данные операции позволили преобразовать полученный датасет в единый объект и использовать его в дальнейшем, не повторяя процедуру получения датасета.

NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.

Matplotlib — это мощная бибилиотека двумерной графики для Python, с помощью которой можно создавать высококачественные рисунки различных форматов. В программе использование данный библиотеки ограничивалось методами plot и show для создания и отображения графиков, представляющих собой зависимость отношения сторон ограничительной рамки от номера кадра. Данные операции были необходимы для визуального анализа полученных и преобразованных данных.

TensorFlow – самая популярная и мощная библиотека Python от Google, написанная на языке программирования C, позволяющая реализовывать алгоритмы машинного обучения и нейронных сетей в частности. В программе использовался модуль TensorFlow Keras для моделирования структуры нейронной сети, обучения, получения результатов. Нейронная сеть была создана на основе класса Sequential, предоставляющего пустую модель нейронной сети, работающей с данными, представленными последовательностями. Далее модель дополнялась слоями с помощью метода add, передавая в качестве параметров какой слой необходимо, например, слой LSTM с размерностью входных данных 300 на 2, где 300 – длина последовательности, а 2 – количество признаков, или Dense, представляющий собой полносвязный слой с суммированием результатов с выходов предыдущего LSTM слоя. А также слой Dropout, позволяющий предотвратить переобучение нейронной сети с помощью отбрасывания входящих весов с какой-либо вероятностью, в программе с вероятностью «0.1». Для получения описания и структурной схемы нейронной сети использовался метод summary, а для сборки и компиляции самой нейронной сети «compile», с переданными функцией ошибки, методом оптимизации и метрикой оценки в качестве параметров. После моделирования структуры нейронной сети, сеть необходимо обучить с помощью метода fit, в который передаются данные для обучения, указывается размер партии для обучения (batch) и количество эпох обучения, а также дополнительно можно передать данные для проверки нейронной сети, так называемые валидационные данные. Для оценки эффективности обученной нейронной сети последовало использовать метод evaluate, передав данные для тестирования и получив значения функции ошибки и значение метрики. Для предсказания конкретного случая служит метод predict, получающий неразмеченные данные на входе и возвращающий результат на выходе 0, если падения не было, и 1, если было.

## **2.3 Блок-схема и описание работы алгоритма**

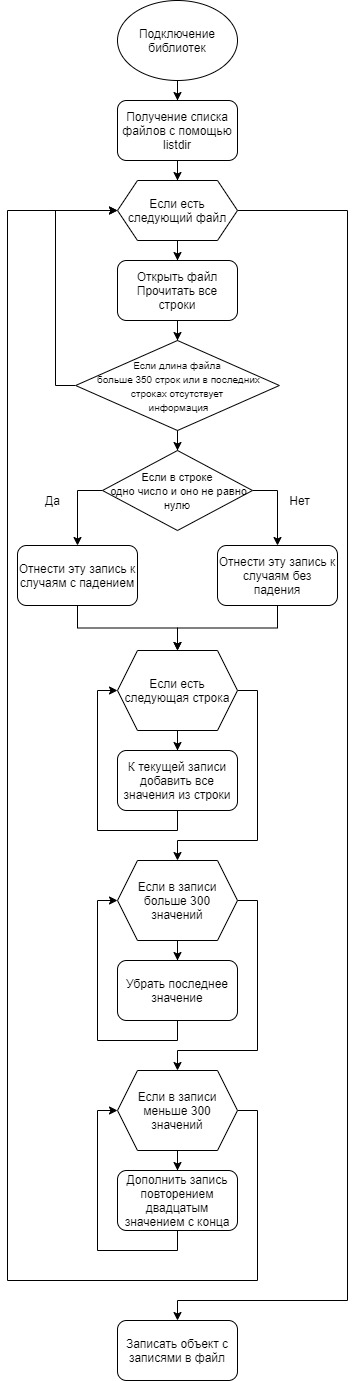


Рисунок - 2.2 Блок-схема получения датасета

Разработку алгоритма необходимо начать с получения данных для обучения нейронной сети. Для этого все аннотационные файлы были перемещены в единую директорию «Annotation files». Вся дальнейшая работа по получению данных из аннотационных файлов производилась в среде PyCharm с использованием стандартного интерпретатора Python.

Так как данные достаточно разнообразные и имеют разную длительность. Визуально было определено, что большинство записей имеет длину около 300. Поэтому было принято решение отобрать данные длинной не более 350.

В аннотационных записях, первые две строки указывают на наличие или отсутствие падения в записи. Таким образом, если первое число в записи равно нулю, значит запись относится к записям без падения. В других случаях падение присутствует.

Так как длины записей разные, следует привести их к единой. Было решено привести записи к длине 300 значений. Более длинные записи урезались до размера 300, а менее длинные дополнялись повторением двадцатого значения с конца, для сохранения динамики в значениях записи.

После определения характера записи и приведения к единому размеру, все значения записи дополняют объект, содержащий в себе значения всех записей. По истечению всех записей, объект со всеми значения экспортируется в файл для дальнейшей работы с данными.



Рисунок - 2.3 Блок-схема обработки данных

Следующей операцией является обработка полученных данных и отбор выборки для обучения и тестирования. Для её выполнения и выполнения всех следующих операций использовалась среда Jupyter с интерпретатором IPython.

Данные импортируются из файлов два объекта x и y для данных со значениями и характером записей соответственно. После импортирования из генеральной выборки были отброшены варианты под номерами: 9, 15, 21, 22, 23, 24, 25, 26, 27, 29, 30, 31, 33, 37, 41, 45, 46, 47, 48, 52, 53, 54, 56, 57, 60, 65, 66, 67, 68, 70, 71, 72, 73, 76, 77, 79, 85, 86.

Далее данные было необходимо разделить данные значений на два массива. Первый, содержащих значения соотношений сторон ограничительной рамки, преобразованные из длин сторон рамки. После преобразования выявились отступления со значениями соотношения большими 10. Такие значения были приравнены нулю. Второй, содержащий скорость перемещения рамки за каждый кадр, преобразованный из координат рамки.

Значения, полученных после всех преобразований, скоростей перемещения рамки за кадр можно увидеть на рисунке – 2.4.

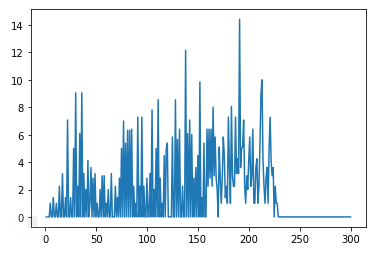
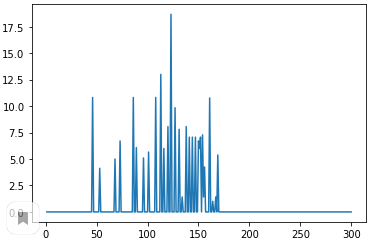
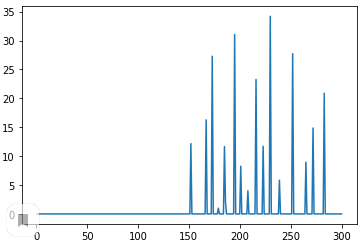


Рисунок - 2.4 Графическое представление значений скорости перемещения за кадр ограничительной рамки

Впоследствии разделения данных необходимо интерполировать значения соотношений сторон рамки в точках, где значения были переопределены как 0, относительно всех значений записи. Эта операция была выполнена методом interp из библиотеки numpy. Графическое представление преобразованных значений соотношений на рисунке - 2.5.

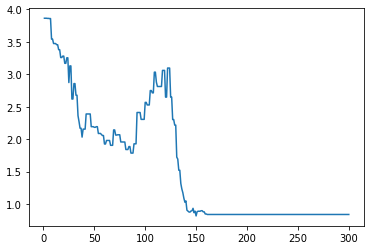
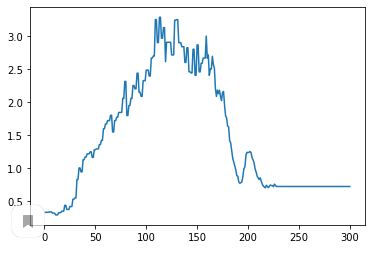
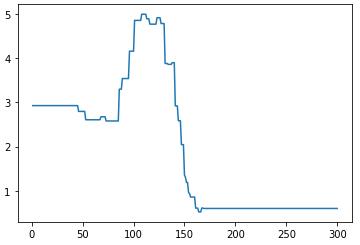


Рисунок - 2.5 Графическое представление значений соотношений сторон ограничительной рамки

Таким образом были получены заполненные данные без пропусков и с минимальным количеством отклонений. Проведя несколько экспериментов было выяснено, что сглаживание значений соотношений сторон с помощью свёртки плохо влияет на обучение нейронной сети. После всех преобразований данные были пронормированы по формуле:

Далее два массива данных необходимо соединить, чтобы получить выборку признаков и вручную был проведён окончательный отбор данных для обучения рекуррентной нейронной сети. Были отброшены варианты 5, 6, 7, 8, 9, 16, 17, 18, 20, 21, 22, 23, 25, 26, 27, 28, 29, 31, 32, 33, 35, 36, 37, 38, 39, 40.

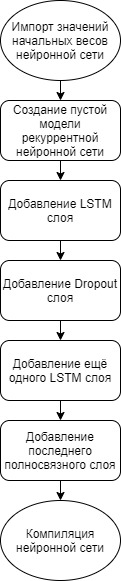


Рисунок - 2.6 Блок-схема моделирования нейронной сети

Затем следует моделирование структуры рекуррентной нейронной сети (Рисунок - 2.6). Проведя некоторое количество экспериментов, были отобраны лучшие случайные начальные веса и сохранены в файл. Перед обучением нейронной сети эти веса импортируются и в каждом слое устанавливаются сохраненные значения весов.

Всё начинается с создания постой рекуррентной нейронной сети на основе класса Sequential из модуля Keras библиотеки TensorFlow.

Первым входным слоем нейронной сети является LSTM слой. Размерность данных слоя 300 на 2, где 300 – длина записи, а 2 – количество признаков, в нашем случае: соотношение сторон и скорость перемещения за кадр. Количество данных на входе было определено 128, как оптимальный вариант. Функция активации «Гиперболический тангенс» была выбрана экспериментально.

Второй слой – Dropout, отбрасывающий значения весов на входе слоя с оптимальной вероятностью 0.1. Таким образом предотвратив переобучение нейронной сети обнаруженной на стадии проведения экспериментов по моделирования нейронной сети.

Третий слой – снова LSTM слой с такой же функцией активации, но количество входных данных сократилось в 4 раза до 32.

Последний слой – Dense – полносвязный слой, суммирующий выходы предыдущего LSTM слоя, активирующий значения с помощью «Сигмоидальной» функции активации.

Компиляция нейронной сети производилась с помощью метода оптимизации «Adam», функцией ошибки «Бинарная кроссэнтропия» и метрикой «Accuracy», определяющей соотношение правильно предсказанных значений ко всем предсказанным значениям. Таким образом метод summary возвращает следующую структуру рекуррентной нейронной сети:

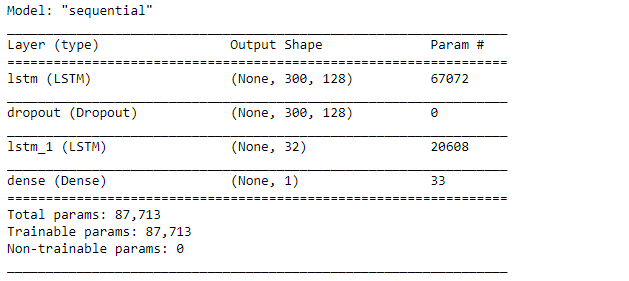


Рисунок - 2.7. Структура нейронной сети.

Последней операцией в решении задачи является обучение модели рекуррентной нейронной сети и получение результатов (Рисунок – 2.8).



Рисунок – 2.8 Блок-схема обучения нейронной сети

Перед обучением нейронной сети необходимо полученную ранее выборку разбить на тренировочную и тестовую подвыборки. Это разбиение было произведено вручную, для получения в подвыборки для тестирования одинаковое соотношение случаев с падением и без падения. В тестовую выборку были отобраны варианты: 19, 24, 30, 44, 45, 46, 47, 48. Таким образом количество тестовых вариантов - 8, а количество тренировочных вариантов – 15.

Первым этапом обучения является обычное обучение модели на основе сформированных подвыборок с размером пакета - 32 и количеством эпох – 60. После первого обучения результат следующий (Рисунок – 2.9-2.10). Среднее значение функции потерь - 1.94, значение метрики – 0.625. Что далеко от хорошего результата. Чем меньше значение функции ошибки и чем ближе значение данной метрики к единице, тем лучше.

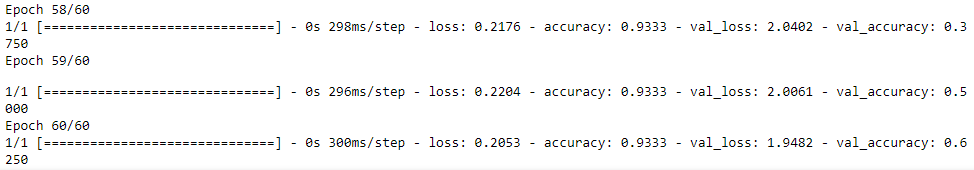


Рисунок - 2.9 Результаты последних трёх эпох перового обучения.

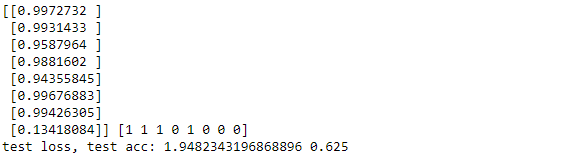


Рисунок - 2.10 Результаты проверки сети после первого обучения с помощью тестовых данных.

Следующим этапом обучения стало обучение на кроссвалидационных данных, полученных с помощью так называемой K-Fold кроссвалидации. Разбивающей выборку на подвыборки следующим образом (Рисунок – 2.11).

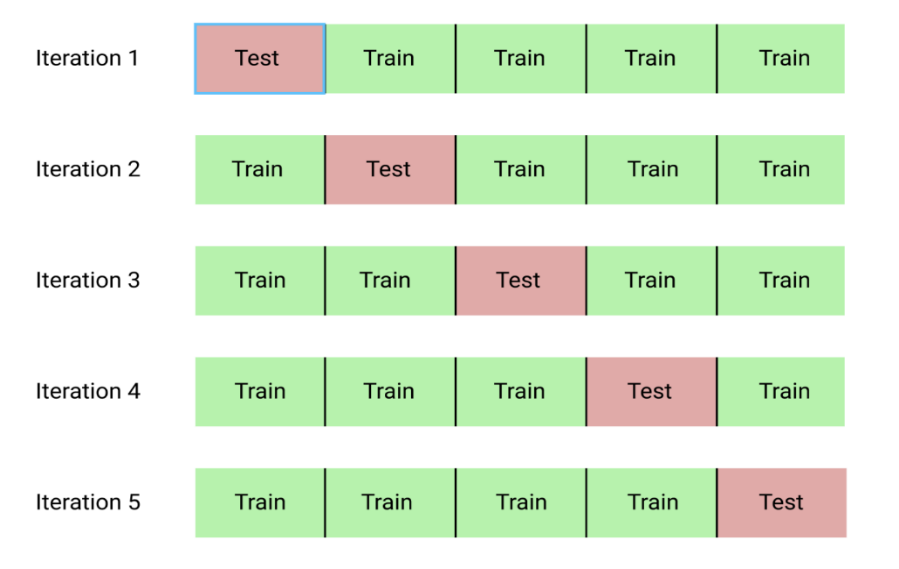


Рисунок – 2.11 Визуальный пример кроссвалидации, (взят с https://www.machinelearningmastery.ru/img/0-752547-79267.png)

Таким образом получилось 3 набора кроссвалидационных подвыборок. Далее нейронная сеть обучалась на этих трёх наборах и результаты обучения в виде значений метрики были следующими:

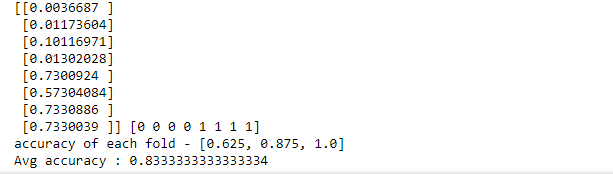


Рисунок - 2.12 Результаты обучения

Из чего видно, что после ещё трёх итераций обучения получилось достичь значение метрики в 0.8(3). А значение функции потерь удалось получить на послей эпохе обучения – 0.2035.

# ЗАКЛЮЧЕНИЕ

Таким образом в ходе выполнения курсового проекта удалось построить модель рекуррентной нейронной сети на основе LSTM слоёв для определения события «падения человека» на видео. Для достижения поставленной цели были выполнены следующие задачи:

1. Изучены методы диагностики состояния человека, выбран инструментарий для реализации;
2. Подготовлен набор данных (датасет) отображающих соответствие параметров измеряемых физических пар-ров состоянию человека.
3. Выполнена предобработка и нормировка данных для обучения и нейронной сети как классификатора, разработано ПО для его реализации
4. Разработана структуру нейронной сети типа LSTM для обучения и классификации состояний человека;
5. Произведено обучение модели классификатора. Получены и проанализированы результаты тестирования.

После обучения нейронной сети на размеченных данных, собранных в Бургундском университете удалось достичь точности в 0.8(3) или 83.3%. Данная точность достаточно высока, но для реального использования модели недостаточна. Для повышения точности следует использовать более чистые и более качественные данные, т.к. из 191 видео только 112 были аннотированы и только 87 прошли проверку на пригодность. Из 87 записей 13 записей – записи «без падения», остальные – записи «с падением». После преобразований данных, записей «без падения» осталось 11. Количество записей «с падением» и «без падения» необходимо было сбалансировать и после чего осталось 23 записи для обучения. Исходя из этого можно предположить, что использование для обучения большего количества данных сможет поправить ситуацию и добиться лучшего результата.

# список Используемых источников

1. Bogdan Kwolek, Michal Kepski, Human fall detection on embedded platform using depth maps and wireless accelerometer, Computer Methods and Programs in Biomedicine, 2014, стр. 489-501

2. LSTM [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/wunderfund/blog/331310/ - Дата доступа: 26.04.2021

3. Датасет Бургундского универститеа [Электронный ресурс]. – Режим доступа: https://imvia.u-bourgogne.fr/en/database/fall-detection-dataset-2.html - Дата доступа 26.04.2021

4. Исследования Жевушского университета [Электронный ресурс]. – Режим доступа: <http://fenix.univ.rzeszow.pl/~mkepski/ds/uf.html> - Дата доступа 26.04.2021

# Приложение

Приложение А - Программный код, выполняющий извлечение данных из файлов аннотации

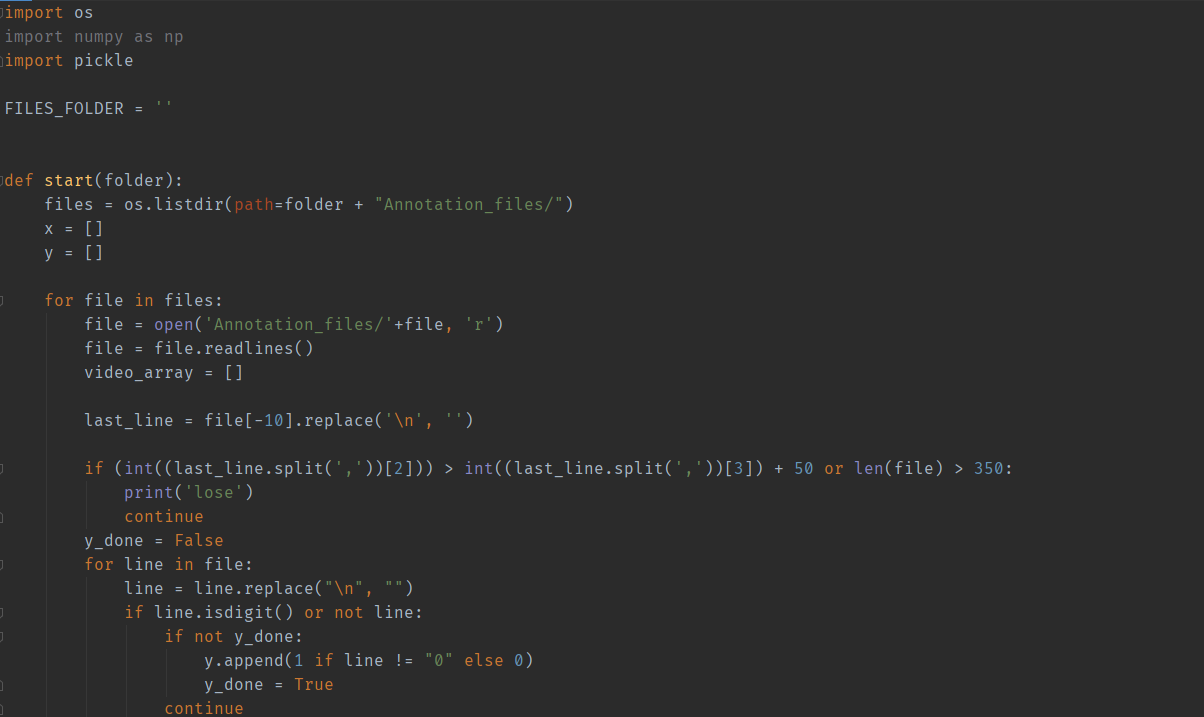


Рисунок – П.1.1

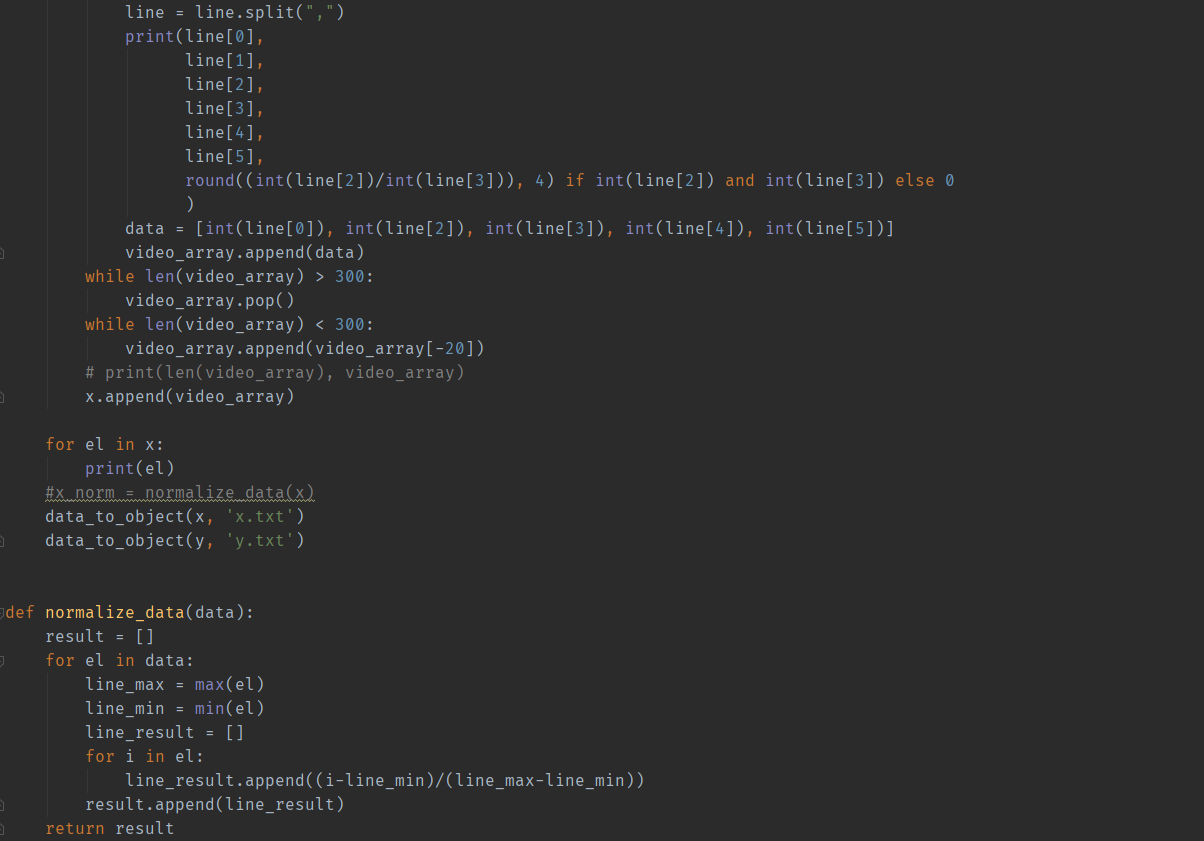


Рисунок – П.1.2

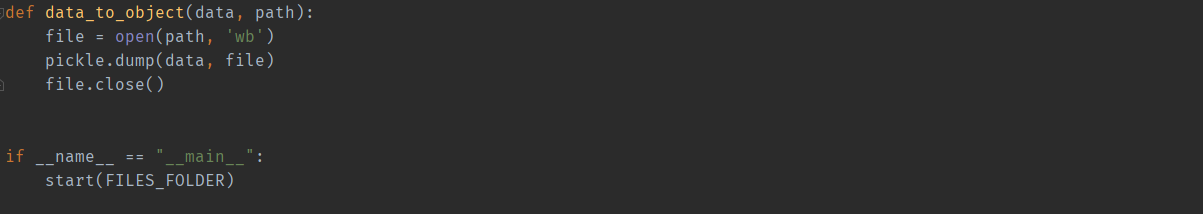


Рисунок – П.1.3

Приложение Б - Программный код, выполняющий обработку данных и обучение нейронной сети:

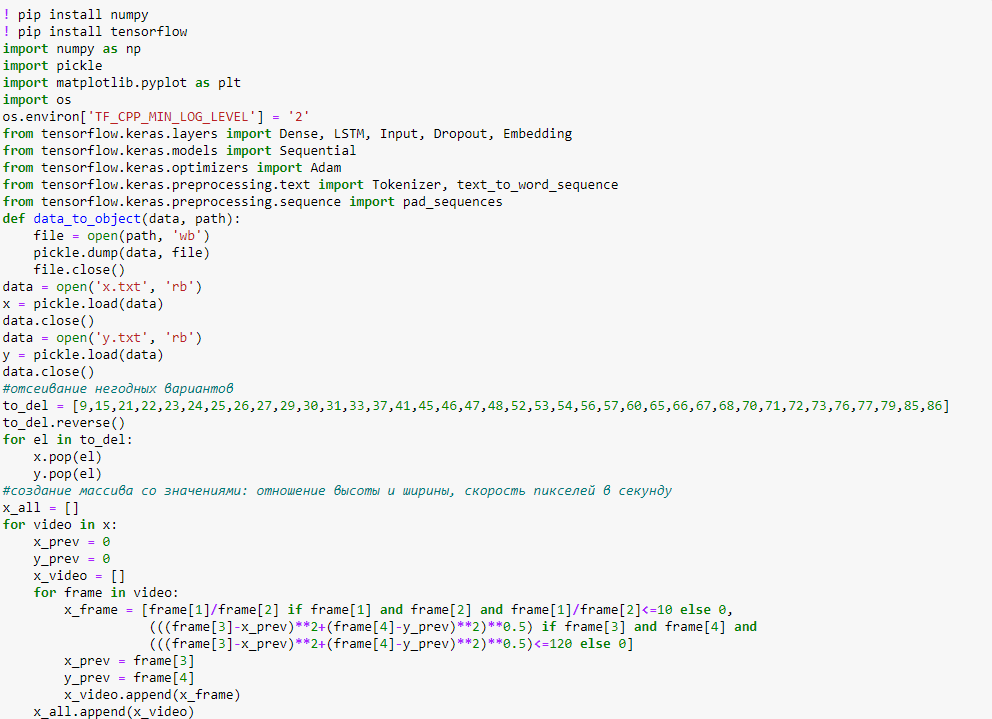


Рисунок – П.2.1

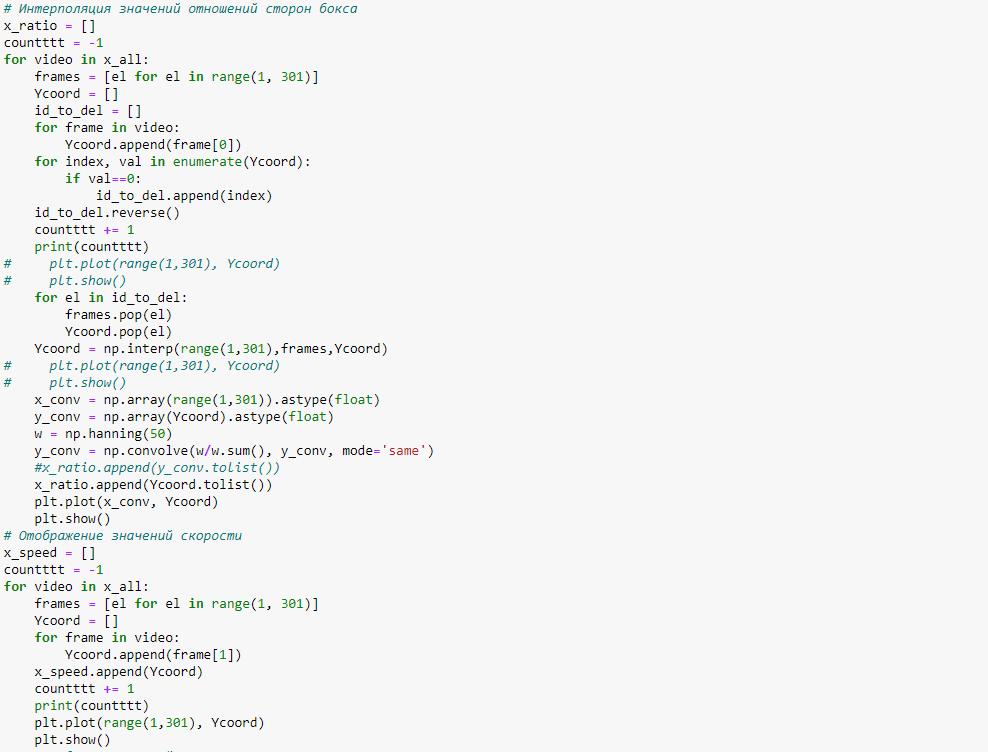


Рисунок – П.2.2

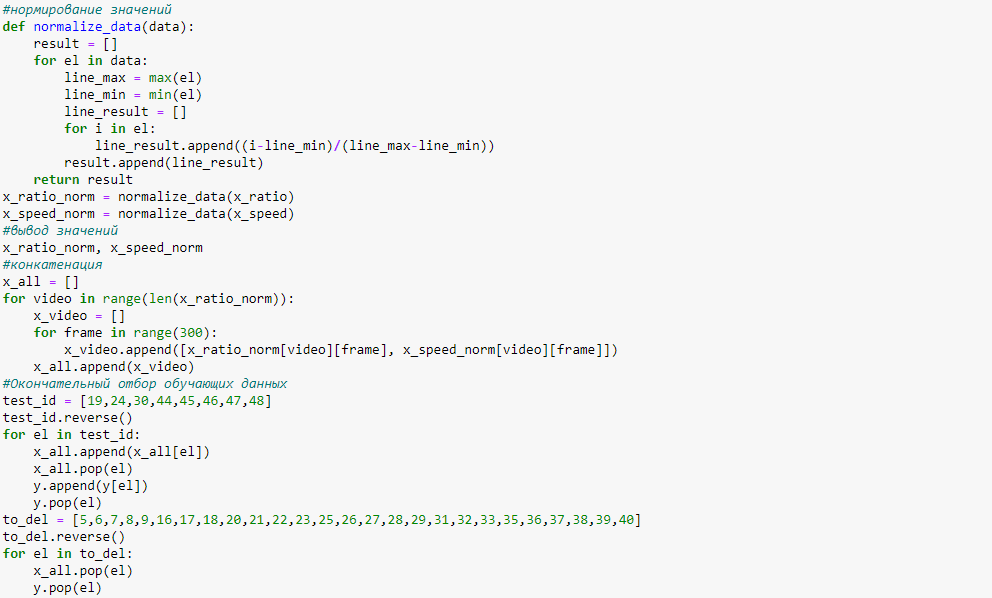


Рисунок – П.2.3



Рисунок – П.2.4

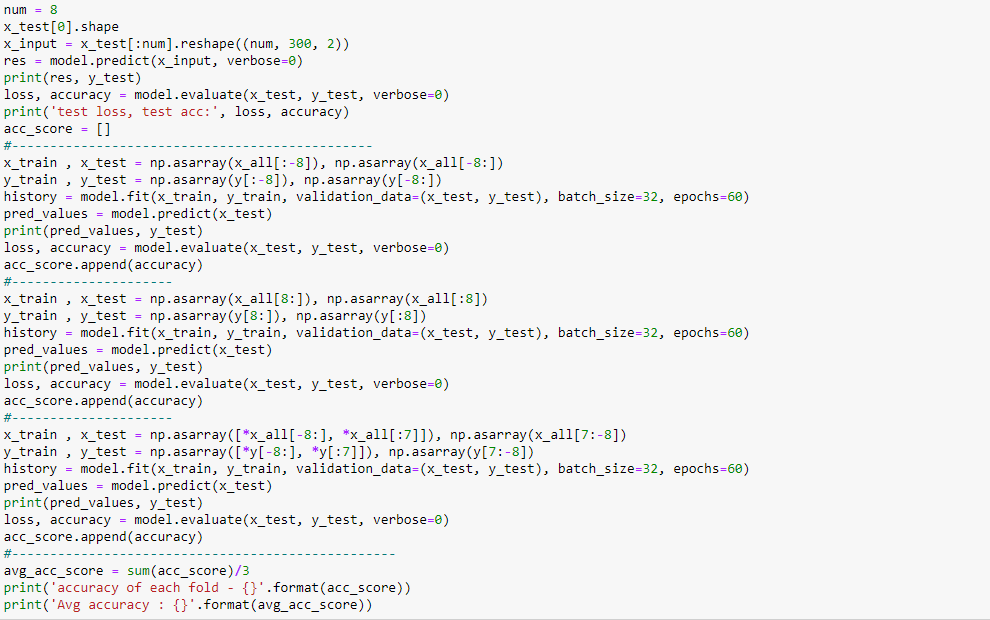
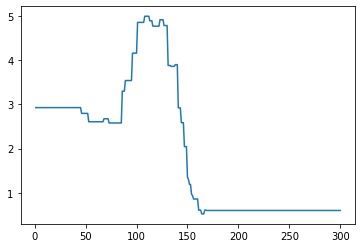


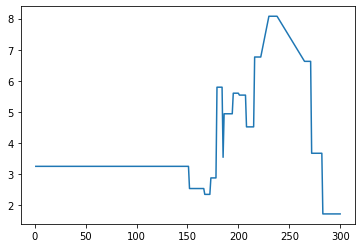
Рисунок – П.2.5

Приложение В - Графики зависимости отношения сторон ограничительной рамки от номера кадра:

0



1



2

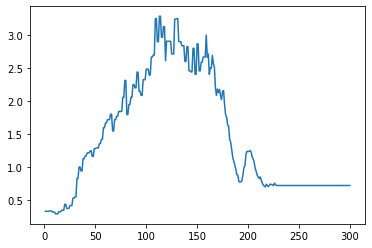
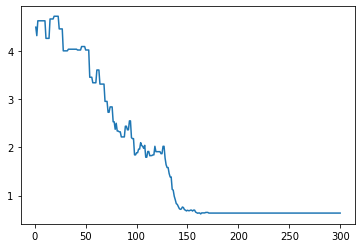
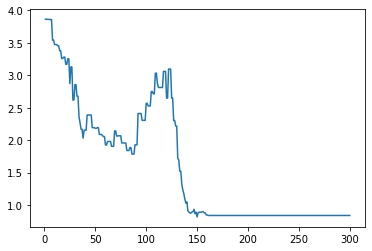


Рисунок – П.3.1

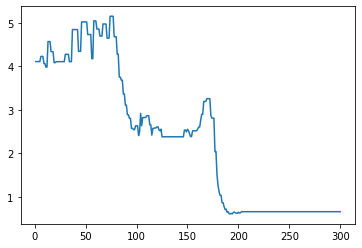
3



4



5



6

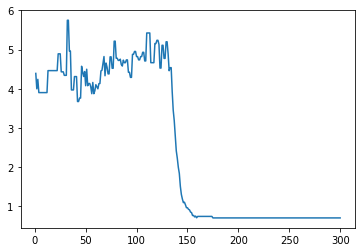
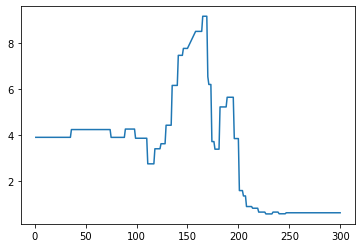
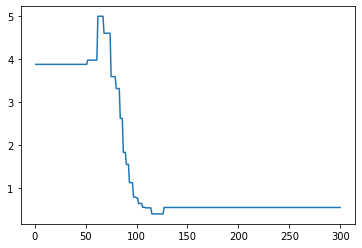


Рисунок – П.3.2

7



8



9

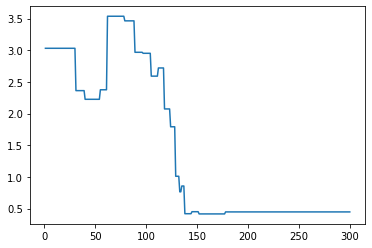
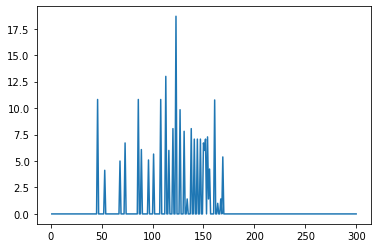


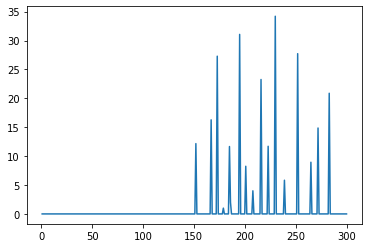
Рисунок – П.3.3

Приложение Г - Графики зависимости скорости перемещения ограничительной рамки от номера кадра:

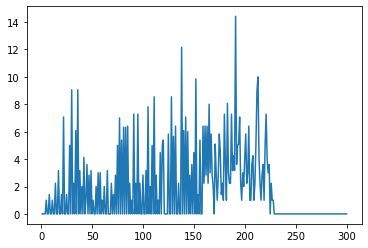
0



1



2



3

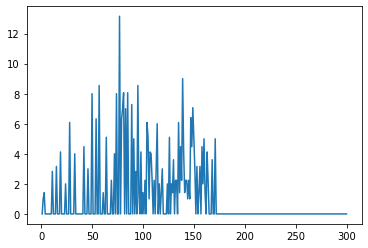
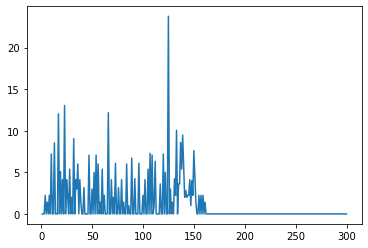
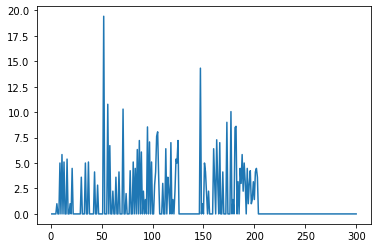


Рисунок – П.4.1

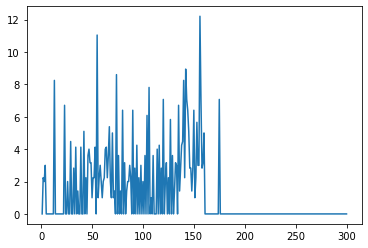
4



5



6



7

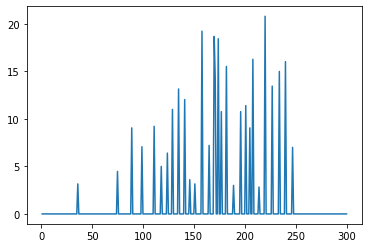
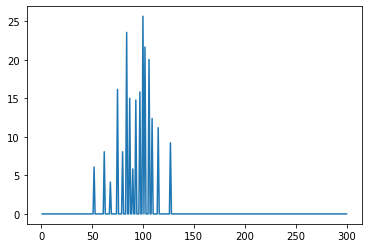


Рисунок – П.4.2

8



9

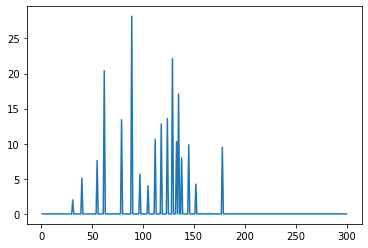


Рисунок – П.4.3

Приложение Д - Результаты первого и последнего этапов обучения:

Epoch 1/60

1/1 [==============================] - 4s 4s/step - loss: 0.6937 - accuracy: 0.5333 - val\_loss: 0.6932 - val\_accuracy: 0.5000

Epoch 2/60

1/1 [==============================] - 0s 305ms/step - loss: 0.6926 - accuracy: 0.5333 - val\_loss: 0.6936 - val\_accuracy: 0.5000

Epoch 3/60

1/1 [==============================] - 0s 308ms/step - loss: 0.6912 - accuracy: 0.5333 - val\_loss: 0.6941 - val\_accuracy: 0.5000

Epoch 4/60

1/1 [==============================] - 0s 304ms/step - loss: 0.6899 - accuracy: 0.5333 - val\_loss: 0.6948 - val\_accuracy: 0.5000

Epoch 5/60

1/1 [==============================] - 0s 402ms/step - loss: 0.6876 - accuracy: 0.5333 - val\_loss: 0.6960 - val\_accuracy: 0.5000

Epoch 6/60

1/1 [==============================] - 0s 308ms/step - loss: 0.6846 - accuracy: 0.5333 - val\_loss: 0.6981 - val\_accuracy: 0.5000

Epoch 7/60

1/1 [==============================] - 0s 303ms/step - loss: 0.6797 - accuracy: 0.6000 - val\_loss: 0.7021 - val\_accuracy: 0.5000

Epoch 8/60

1/1 [==============================] - 0s 305ms/step - loss: 0.6710 - accuracy: 0.6000 - val\_loss: 0.7100 - val\_accuracy: 0.5000

Epoch 9/60

1/1 [==============================] - 0s 295ms/step - loss: 0.6553 - accuracy: 0.5333 - val\_loss: 0.7195 - val\_accuracy: 0.5000

Epoch 10/60

1/1 [==============================] - 0s 318ms/step - loss: 0.6172 - accuracy: 0.5333 - val\_loss: 0.6917 - val\_accuracy: 0.5000

Epoch 11/60

1/1 [==============================] - 0s 301ms/step - loss: 0.6078 - accuracy: 0.6000 - val\_loss: 0.7220 - val\_accuracy: 0.5000

Epoch 12/60

1/1 [==============================] - 0s 298ms/step - loss: 0.5696 - accuracy: 0.7333 - val\_loss: 0.7430 - val\_accuracy: 0.5000

Epoch 13/60

1/1 [==============================] - 0s 298ms/step - loss: 0.6030 - accuracy: 0.5333 - val\_loss: 0.7542 - val\_accuracy: 0.5000

Epoch 14/60

1/1 [==============================] - 0s 301ms/step - loss: 0.6140 - accuracy: 0.5333 - val\_loss: 0.7646 - val\_accuracy: 0.5000

Epoch 15/60

1/1 [==============================] - 0s 297ms/step - loss: 0.6197 - accuracy: 0.5333 - val\_loss: 0.7754 - val\_accuracy: 0.5000

Epoch 16/60

1/1 [==============================] - 0s 301ms/step - loss: 0.6203 - accuracy: 0.5333 - val\_loss: 0.7864 - val\_accuracy: 0.5000

Epoch 17/60

1/1 [==============================] - 0s 294ms/step - loss: 0.6166 - accuracy: 0.5333 - val\_loss: 0.7972 - val\_accuracy: 0.5000

Epoch 18/60

1/1 [==============================] - 0s 293ms/step - loss: 0.6108 - accuracy: 0.5333 - val\_loss: 0.8076 - val\_accuracy: 0.5000

Epoch 19/60

1/1 [==============================] - 0s 306ms/step - loss: 0.5966 - accuracy: 0.5333 - val\_loss: 0.8181 - val\_accuracy: 0.5000

Epoch 20/60

1/1 [==============================] - 0s 312ms/step - loss: 0.5734 - accuracy: 0.6000 - val\_loss: 0.8287 - val\_accuracy: 0.5000

Epoch 21/60

1/1 [==============================] - 0s 297ms/step - loss: 0.5352 - accuracy: 0.8000 - val\_loss: 0.8349 - val\_accuracy: 0.5000

Epoch 22/60

1/1 [==============================] - 0s 297ms/step - loss: 0.4770 - accuracy: 0.8667 - val\_loss: 0.8197 - val\_accuracy: 0.5000

Epoch 23/60

1/1 [==============================] - 0s 305ms/step - loss: 0.4303 - accuracy: 0.8667 - val\_loss: 0.7884 - val\_accuracy: 0.5000

Epoch 24/60

1/1 [==============================] - 0s 298ms/step - loss: 0.4998 - accuracy: 0.7333 - val\_loss: 0.9724 - val\_accuracy: 0.5000

Epoch 25/60

1/1 [==============================] - 0s 298ms/step - loss: 0.3644 - accuracy: 0.8667 - val\_loss: 1.1369 - val\_accuracy: 0.5000

Epoch 26/60

1/1 [==============================] - 0s 293ms/step - loss: 0.3804 - accuracy: 0.8667 - val\_loss: 1.2573 - val\_accuracy: 0.5000

Epoch 27/60

1/1 [==============================] - 0s 299ms/step - loss: 0.4090 - accuracy: 0.8667 - val\_loss: 1.3177 - val\_accuracy: 0.5000

Epoch 28/60

1/1 [==============================] - 0s 307ms/step - loss: 0.3785 - accuracy: 0.8667 - val\_loss: 1.3272 - val\_accuracy: 0.5000

Epoch 29/60

1/1 [==============================] - 0s 299ms/step - loss: 0.2885 - accuracy: 0.9333 - val\_loss: 1.0940 - val\_accuracy: 0.5000

Epoch 30/60

1/1 [==============================] - 0s 299ms/step - loss: 0.4633 - accuracy: 0.7333 - val\_loss: 1.4565 - val\_accuracy: 0.6250

Epoch 31/60

1/1 [==============================] - 0s 298ms/step - loss: 0.2910 - accuracy: 0.8667 - val\_loss: 1.5558 - val\_accuracy: 0.5000

Epoch 32/60

1/1 [==============================] - 0s 300ms/step - loss: 0.4000 - accuracy: 0.8667 - val\_loss: 1.5935 - val\_accuracy: 0.5000

Epoch 33/60

1/1 [==============================] - 0s 302ms/step - loss: 0.4077 - accuracy: 0.8667 - val\_loss: 1.5546 - val\_accuracy: 0.6250

Epoch 34/60

1/1 [==============================] - 0s 300ms/step - loss: 0.3360 - accuracy: 0.8667 - val\_loss: 1.4954 - val\_accuracy: 0.3750

Epoch 35/60

1/1 [==============================] - 0s 299ms/step - loss: 0.2724 - accuracy: 0.9333 - val\_loss: 1.2773 - val\_accuracy: 0.5000

Epoch 36/60

1/1 [==============================] - 0s 310ms/step - loss: 0.4279 - accuracy: 0.7333 - val\_loss: 1.6147 - val\_accuracy: 0.5000

Epoch 37/60

1/1 [==============================] - 0s 297ms/step - loss: 0.2548 - accuracy: 0.9333 - val\_loss: 1.6190 - val\_accuracy: 0.6250

Epoch 38/60

1/1 [==============================] - 0s 302ms/step - loss: 0.3580 - accuracy: 0.8667 - val\_loss: 1.6380 - val\_accuracy: 0.6250

Epoch 39/60

1/1 [==============================] - 0s 301ms/step - loss: 0.3820 - accuracy: 0.8667 - val\_loss: 1.6051 - val\_accuracy: 0.6250

Epoch 40/60

1/1 [==============================] - 0s 301ms/step - loss: 0.3286 - accuracy: 0.8667 - val\_loss: 1.6348 - val\_accuracy: 0.5000

Epoch 41/60

1/1 [==============================] - 0s 294ms/step - loss: 0.2476 - accuracy: 0.9333 - val\_loss: 1.4362 - val\_accuracy: 0.3750

Epoch 42/60

1/1 [==============================] - 0s 296ms/step - loss: 0.3078 - accuracy: 0.8667 - val\_loss: 1.4183 - val\_accuracy: 0.3750

Epoch 43/60

1/1 [==============================] - 0s 294ms/step - loss: 0.3205 - accuracy: 0.8667 - val\_loss: 1.6426 - val\_accuracy: 0.5000

Epoch 44/60

1/1 [==============================] - 0s 298ms/step - loss: 0.2448 - accuracy: 0.9333 - val\_loss: 1.6141 - val\_accuracy: 0.6250

Epoch 45/60

1/1 [==============================] - 0s 299ms/step - loss: 0.2713 - accuracy: 0.8667 - val\_loss: 1.6172 - val\_accuracy: 0.6250

Epoch 46/60

1/1 [==============================] - 0s 299ms/step - loss: 0.3110 - accuracy: 0.8667 - val\_loss: 1.6181 - val\_accuracy: 0.6250

Epoch 47/60

1/1 [==============================] - 0s 305ms/step - loss: 0.2778 - accuracy: 0.8667 - val\_loss: 1.6876 - val\_accuracy: 0.5000

Epoch 48/60

1/1 [==============================] - 0s 296ms/step - loss: 0.2376 - accuracy: 0.9333 - val\_loss: 1.6867 - val\_accuracy: 0.3750

Epoch 49/60

1/1 [==============================] - 0s 300ms/step - loss: 0.2502 - accuracy: 0.9333 - val\_loss: 1.6301 - val\_accuracy: 0.3750

Epoch 50/60

1/1 [==============================] - 0s 299ms/step - loss: 0.2717 - accuracy: 0.8667 - val\_loss: 1.7067 - val\_accuracy: 0.3750

Epoch 51/60

1/1 [==============================] - 0s 295ms/step - loss: 0.2571 - accuracy: 0.9333 - val\_loss: 1.7858 - val\_accuracy: 0.5000

Epoch 52/60

1/1 [==============================] - 0s 301ms/step - loss: 0.2265 - accuracy: 0.9333 - val\_loss: 1.7471 - val\_accuracy: 0.6250

Epoch 53/60

1/1 [==============================] - 0s 299ms/step - loss: 0.2314 - accuracy: 0.9333 - val\_loss: 1.7411 - val\_accuracy: 0.6250

Epoch 54/60

1/1 [==============================] - 0s 300ms/step - loss: 0.2406 - accuracy: 0.9333 - val\_loss: 1.7667 - val\_accuracy: 0.6250

Epoch 55/60

1/1 [==============================] - 0s 297ms/step - loss: 0.2387 - accuracy: 0.9333 - val\_loss: 1.8008 - val\_accuracy: 0.6250

Epoch 56/60

1/1 [==============================] - 0s 306ms/step - loss: 0.2230 - accuracy: 0.9333 - val\_loss: 1.9078 - val\_accuracy: 0.5000

Epoch 57/60

1/1 [==============================] - 0s 302ms/step - loss: 0.2130 - accuracy: 0.9333 - val\_loss: 1.9952 - val\_accuracy: 0.5000

Epoch 58/60

1/1 [==============================] - 0s 298ms/step - loss: 0.2176 - accuracy: 0.9333 - val\_loss: 2.0402 - val\_accuracy: 0.3750

Epoch 59/60

1/1 [==============================] - 0s 296ms/step - loss: 0.2204 - accuracy: 0.9333 - val\_loss: 2.0061 - val\_accuracy: 0.5000

Epoch 60/60

1/1 [==============================] - 0s 300ms/step - loss: 0.2053 - accuracy: 0.9333 - val\_loss: 1.9482 - val\_accuracy: 0.6250

Epoch 1/60

1/1 [==============================] - 0s 363ms/step - loss: 0.4983 - accuracy: 0.7333 - val\_loss: 0.1844 - val\_accuracy: 1.0000

Epoch 2/60

1/1 [==============================] - 0s 296ms/step - loss: 0.4959 - accuracy: 0.7333 - val\_loss: 0.1833 - val\_accuracy: 1.0000

Epoch 3/60

1/1 [==============================] - 0s 300ms/step - loss: 0.4942 - accuracy: 0.7333 - val\_loss: 0.1809 - val\_accuracy: 1.0000

Epoch 4/60

1/1 [==============================] - 0s 298ms/step - loss: 0.4938 - accuracy: 0.7333 - val\_loss: 0.1791 - val\_accuracy: 1.0000

Epoch 5/60

1/1 [==============================] - 0s 302ms/step - loss: 0.4941 - accuracy: 0.7333 - val\_loss: 0.1786 - val\_accuracy: 1.0000

Epoch 6/60

1/1 [==============================] - 0s 304ms/step - loss: 0.4936 - accuracy: 0.7333 - val\_loss: 0.1789 - val\_accuracy: 1.0000

Epoch 7/60

1/1 [==============================] - 0s 295ms/step - loss: 0.4918 - accuracy: 0.7333 - val\_loss: 0.1796 - val\_accuracy: 1.0000

Epoch 8/60

1/1 [==============================] - 0s 304ms/step - loss: 0.4925 - accuracy: 0.7333 - val\_loss: 0.1806 - val\_accuracy: 1.0000

Epoch 9/60

1/1 [==============================] - 0s 294ms/step - loss: 0.4913 - accuracy: 0.7333 - val\_loss: 0.1818 - val\_accuracy: 1.0000

Epoch 10/60

1/1 [==============================] - 0s 296ms/step - loss: 0.4922 - accuracy: 0.7333 - val\_loss: 0.1832 - val\_accuracy: 1.0000

Epoch 11/60

1/1 [==============================] - 0s 306ms/step - loss: 0.4913 - accuracy: 0.7333 - val\_loss: 0.1847 - val\_accuracy: 1.0000

Epoch 12/60

1/1 [==============================] - 0s 303ms/step - loss: 0.4907 - accuracy: 0.7333 - val\_loss: 0.1863 - val\_accuracy: 1.0000

Epoch 13/60

1/1 [==============================] - 0s 336ms/step - loss: 0.4907 - accuracy: 0.7333 - val\_loss: 0.1882 - val\_accuracy: 1.0000

Epoch 14/60

1/1 [==============================] - 0s 297ms/step - loss: 0.4891 - accuracy: 0.7333 - val\_loss: 0.1904 - val\_accuracy: 1.0000

Epoch 15/60

1/1 [==============================] - 0s 303ms/step - loss: 0.4911 - accuracy: 0.7333 - val\_loss: 0.1927 - val\_accuracy: 1.0000

Epoch 16/60

1/1 [==============================] - 0s 307ms/step - loss: 0.4888 - accuracy: 0.7333 - val\_loss: 0.1947 - val\_accuracy: 1.0000

Epoch 17/60

1/1 [==============================] - 0s 298ms/step - loss: 0.4885 - accuracy: 0.7333 - val\_loss: 0.1961 - val\_accuracy: 1.0000

Epoch 18/60

1/1 [==============================] - 0s 309ms/step - loss: 0.4885 - accuracy: 0.7333 - val\_loss: 0.1971 - val\_accuracy: 1.0000

Epoch 19/60

1/1 [==============================] - 0s 300ms/step - loss: 0.4899 - accuracy: 0.7333 - val\_loss: 0.1978 - val\_accuracy: 1.0000

Epoch 20/60

1/1 [==============================] - 0s 301ms/step - loss: 0.4881 - accuracy: 0.7333 - val\_loss: 0.1984 - val\_accuracy: 1.0000

Epoch 21/60

1/1 [==============================] - 0s 304ms/step - loss: 0.4890 - accuracy: 0.7333 - val\_loss: 0.1989 - val\_accuracy: 1.0000

Epoch 22/60

1/1 [==============================] - 0s 306ms/step - loss: 0.4880 - accuracy: 0.7333 - val\_loss: 0.1993 - val\_accuracy: 1.0000

Epoch 23/60

1/1 [==============================] - 0s 308ms/step - loss: 0.4873 - accuracy: 0.7333 - val\_loss: 0.1998 - val\_accuracy: 1.0000

Epoch 24/60

1/1 [==============================] - 0s 306ms/step - loss: 0.4879 - accuracy: 0.7333 - val\_loss: 0.2005 - val\_accuracy: 1.0000

Epoch 25/60

1/1 [==============================] - 0s 301ms/step - loss: 0.4875 - accuracy: 0.7333 - val\_loss: 0.2013 - val\_accuracy: 1.0000

Epoch 26/60

1/1 [==============================] - 0s 300ms/step - loss: 0.4871 - accuracy: 0.7333 - val\_loss: 0.2021 - val\_accuracy: 1.0000

Epoch 27/60

1/1 [==============================] - 0s 303ms/step - loss: 0.4882 - accuracy: 0.7333 - val\_loss: 0.2028 - val\_accuracy: 1.0000

Epoch 28/60

1/1 [==============================] - 0s 301ms/step - loss: 0.4880 - accuracy: 0.7333 - val\_loss: 0.2034 - val\_accuracy: 1.0000

Epoch 29/60

1/1 [==============================] - 0s 307ms/step - loss: 0.4865 - accuracy: 0.7333 - val\_loss: 0.2042 - val\_accuracy: 1.0000

Epoch 30/60

1/1 [==============================] - 0s 304ms/step - loss: 0.4850 - accuracy: 0.7333 - val\_loss: 0.2048 - val\_accuracy: 1.0000

Epoch 31/60

1/1 [==============================] - 0s 306ms/step - loss: 0.4856 - accuracy: 0.7333 - val\_loss: 0.2051 - val\_accuracy: 1.0000

Epoch 32/60

1/1 [==============================] - 0s 302ms/step - loss: 0.4877 - accuracy: 0.7333 - val\_loss: 0.2054 - val\_accuracy: 1.0000

Epoch 33/60

1/1 [==============================] - 0s 305ms/step - loss: 0.4856 - accuracy: 0.7333 - val\_loss: 0.2057 - val\_accuracy: 1.0000

Epoch 34/60

1/1 [==============================] - 0s 290ms/step - loss: 0.4852 - accuracy: 0.7333 - val\_loss: 0.2059 - val\_accuracy: 1.0000

Epoch 35/60

1/1 [==============================] - 0s 299ms/step - loss: 0.4858 - accuracy: 0.7333 - val\_loss: 0.2060 - val\_accuracy: 1.0000

Epoch 36/60

1/1 [==============================] - 0s 303ms/step - loss: 0.4854 - accuracy: 0.7333 - val\_loss: 0.2062 - val\_accuracy: 1.0000

Epoch 37/60

1/1 [==============================] - 0s 293ms/step - loss: 0.4862 - accuracy: 0.7333 - val\_loss: 0.2063 - val\_accuracy: 1.0000

Epoch 38/60

1/1 [==============================] - 0s 292ms/step - loss: 0.4846 - accuracy: 0.7333 - val\_loss: 0.2063 - val\_accuracy: 1.0000

Epoch 39/60

1/1 [==============================] - 0s 293ms/step - loss: 0.4849 - accuracy: 0.7333 - val\_loss: 0.2063 - val\_accuracy: 1.0000

Epoch 40/60

1/1 [==============================] - 0s 292ms/step - loss: 0.4847 - accuracy: 0.7333 - val\_loss: 0.2062 - val\_accuracy: 1.0000

Epoch 41/60

1/1 [==============================] - 0s 294ms/step - loss: 0.4846 - accuracy: 0.7333 - val\_loss: 0.2059 - val\_accuracy: 1.0000

Epoch 42/60

1/1 [==============================] - 0s 295ms/step - loss: 0.4835 - accuracy: 0.7333 - val\_loss: 0.2056 - val\_accuracy: 1.0000

Epoch 43/60

1/1 [==============================] - 0s 393ms/step - loss: 0.4857 - accuracy: 0.7333 - val\_loss: 0.2057 - val\_accuracy: 1.0000

Epoch 44/60

1/1 [==============================] - 0s 294ms/step - loss: 0.4840 - accuracy: 0.7333 - val\_loss: 0.2059 - val\_accuracy: 1.0000

Epoch 45/60

1/1 [==============================] - 0s 296ms/step - loss: 0.4841 - accuracy: 0.7333 - val\_loss: 0.2060 - val\_accuracy: 1.0000

Epoch 46/60

1/1 [==============================] - 0s 308ms/step - loss: 0.4832 - accuracy: 0.7333 - val\_loss: 0.2059 - val\_accuracy: 1.0000

Epoch 47/60

1/1 [==============================] - 0s 298ms/step - loss: 0.4841 - accuracy: 0.7333 - val\_loss: 0.2058 - val\_accuracy: 1.0000

Epoch 48/60

1/1 [==============================] - 0s 301ms/step - loss: 0.4823 - accuracy: 0.7333 - val\_loss: 0.2057 - val\_accuracy: 1.0000

Epoch 49/60

1/1 [==============================] - 0s 312ms/step - loss: 0.4839 - accuracy: 0.7333 - val\_loss: 0.2055 - val\_accuracy: 1.0000

Epoch 50/60

1/1 [==============================] - 0s 303ms/step - loss: 0.4828 - accuracy: 0.7333 - val\_loss: 0.2054 - val\_accuracy: 1.0000

Epoch 51/60

1/1 [==============================] - 0s 309ms/step - loss: 0.4829 - accuracy: 0.7333 - val\_loss: 0.2052 - val\_accuracy: 1.0000

Epoch 52/60

1/1 [==============================] - 0s 309ms/step - loss: 0.4816 - accuracy: 0.7333 - val\_loss: 0.2051 - val\_accuracy: 1.0000

Epoch 53/60

1/1 [==============================] - 0s 300ms/step - loss: 0.4831 - accuracy: 0.7333 - val\_loss: 0.2049 - val\_accuracy: 1.0000

Epoch 54/60

1/1 [==============================] - 0s 300ms/step - loss: 0.4823 - accuracy: 0.7333 - val\_loss: 0.2047 - val\_accuracy: 1.0000

Epoch 55/60

1/1 [==============================] - 0s 300ms/step - loss: 0.4824 - accuracy: 0.7333 - val\_loss: 0.2045 - val\_accuracy: 1.0000

Epoch 56/60

1/1 [==============================] - 0s 303ms/step - loss: 0.4825 - accuracy: 0.7333 - val\_loss: 0.2043 - val\_accuracy: 1.0000

Epoch 57/60

1/1 [==============================] - 0s 308ms/step - loss: 0.4797 - accuracy: 0.7333 - val\_loss: 0.2040 - val\_accuracy: 1.0000

Epoch 58/60

1/1 [==============================] - 0s 298ms/step - loss: 0.4804 - accuracy: 0.7333 - val\_loss: 0.2036 - val\_accuracy: 1.0000

Epoch 59/60

1/1 [==============================] - 0s 315ms/step - loss: 0.4819 - accuracy: 0.7333 - val\_loss: 0.2034 - val\_accuracy: 1.0000

Epoch 60/60

1/1 [==============================] - 0s 291ms/step - loss: 0.4821 - accuracy: 0.7333 - val\_loss: 0.2035 - val\_accuracy: 1.0000

[[0.0036687 ]

[0.01173604]

[0.10116971]

[0.01302028]

[0.7300924 ]

[0.57304084]

[0.7330886 ]

[0.7330039 ]] [0 0 0 0 1 1 1 1]

accuracy of each fold - [0.625, 0.875, 1.0]

Avg accuracy : 0.8333333333333334

Приложение Е - Блок-схемы алгоритма

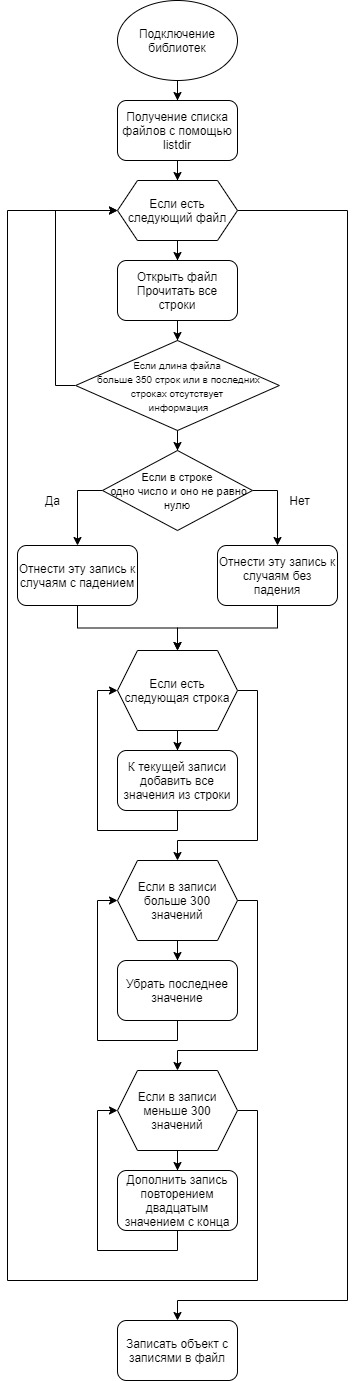
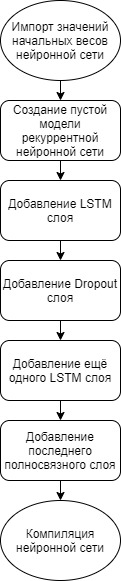
   

Рисунок – П.6.1