

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии.

Работа допущена к защите

Директор ВШПИ

_____ П.Д.Дробинцев

« ____ » _____ 2020 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
работа бакалавра

СИСТЕМА РАСПОЗНАВАНИЯ БОЛЕЗНЕЙ
РАСТЕНИЙ ПО МНОГОМЕРНЫМ ОПИСАНИЯМ
RGB ИЗОБРАЖЕНИЙ ЛИСТЬЕВ

По направлению подготовки (специальности) 09.03.04 – Программная инженерия

Направленность (профиль) 09.03.04_01 – Технология разработки и сопровождения качественного программного продукта

Выполнил

студент гр.3530904/60105

В.Т. Чинь

Руководитель

к.т.н.,

доцент

В.С. Тутыгин

Консультант

по нормоконтролю

Е.Г. Локшина

Санкт-Петербург
2020

**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО
ИКНТ/Высшая школа программной инженерии**

УТВЕРЖДАЮ
Директор ВШПИ

П.Д.Дробинцев
« » 2020г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студенту Чинь Ван Тхань 3530904/60105

фамилия, имя, отчество (при наличии), номер группы

1. Тема работы: Система распознавания болезней растений по многомерным описаниям RGB изображений листьев.
2. Срок сдачи студентом законченной работы:

-
3. Исходные данные по работе:

Имеется массив изображений листьев пшеницы для 8 различных болезней. Требуется разработать алгоритм и программное обеспечение для определения вида заболевания путем цифровой обработки целевого изображения листа пшеницы с признаками болезни.

4. Содержание работы (перечень подлежащих разработке вопросов):

1. Анализ известных методов диагностики болезней растений по изображениям листьев.

- 1.1 На основе нейронной сети.

- 1.2 С использованием текстурного анализа и нечеткой логики.

2. Алгоритм и программная реализация системы

распознавания болезней растений по изображениям листьев.

3. Результат моделирования разработанной системы распознавания.

4. Экспериментальные результаты работы системы распознавания на реальных изображениях листьев пшеницы.

5. Перечень графического материала (с указанием обязательных чертежей):

6. Консультанты по работе: _____

7. Дата выдачи задания _____

Руководитель ВКР

(подпись)

В. С. Тутыгин

инициалы, фамилия

Задание принял к исполнению _____

(дата)

Студент

(подпись)

В. Т. Чинь

инициалы, фамилия

РЕФЕРАТ

На 74 с., 30 рисунков, 4 таблицы, 4 приложений.

КЛЮЧЕВЫЕ СЛОВА: ОБРАБОТКА ИЗОБРАЖЕНИЙ, ТЕКСТУРНЫЙ АНАЛИЗ ИЗОБРАЖЕНИЯ, МАТРИЦА GLCM, ПАРАМЕТРЫ ХАРАЛИКА, НЕЙРОННОЙ СЕТИ, НЕЧЕТКАЯ ЛОГИКА.

Тема выпускной квалификационной работы: «Система распознавания болезней растений по многомерным описаниям RGB изображений листьев».

Данная работа посвящена разработке системы обнаружения и классификации болезней растений по многомерным описаниям RGB изображений листьев. Задачи, которые решались в ходе работы:

1. Разработка метода нормализации фотоизображений листьев.
2. Вычисление текстурных характеристик изображений – характеристики Харалика.
3. Разработка алгоритма классификации болезней растений на основе нечеткой логики.
4. Разработка алгоритма классификации болезней растений на основе нейронной сети.
5. Программная реализация выше алгоритма и метода.

Объект исследования является набором изображений листьев пшеницы.

В работе, рассмотрены существенные методы диагностики болезней растений по изображений, как методы на основе нейронной сети, и метод с использованием текстурного анализа и нечеткой логики. На наше наборе данных, выполняли методом анализа текстурных характеристик чтобы выделяли характеристики для классификации типа заболевания. Классификация реализуется методом нечеткой логики и методом

полностью подключенной сети, и потом сравниваются результаты двух методов. Перед вычислением текстурных характеристик изображений, применяли классический метод обработки изображений, используя открытой библиотекой OpenCV, чтобы удалять из изображения шумы – области ненужной информации. Обработки изображений помогает повысить качество ввода в систему, независимо от того, какой метод диагностики используется.

В результате работы, разработана система распознавания болезней растений по изображениям листьев, двумя методами: метод нечеткой логики и метод нейронной сети. Так же сравнивать результат моделирования работы двух методов, получилось что с небольшим набором данных, метод нечеткой логики достигается точность более 95%. А метод нейронной сети применяется только при большом наборе данных на обучении. Разработанная система нами были проверены на наборе данных пшеницы, но так же предлагаем применить для других типов растений, заболевания которых отображаются на своих листьях, например кукуруза....

ABSTRACT

74 pages, 30 figures, 4 tables, 4 appendices.

KEYWORDS: IMAGE PROCESSING, TEXTURE ANALYSIS OF THE IMAGE, MATRIX GLCM, HARALIK'S FEATURES, DEEP LEARNING, FUZZY LOGIC.

The subject of the graduate qualification work is: "A system for recognizing plant diseases from multidimensional descriptions of RGB leaf images."

This work is devoted to the development of a system for the detection and classification of plant diseases according to multidimensional descriptions of RGB leaf images. Tasks that were solved in the course of work:

1. Development of a method for normalizing leaf images.
2. Calculation of the texture features of images - Haralick's features.
3. Development of an algorithm for classifying plant diseases based on fuzzy logic.
4. Development of an algorithm for classifying plant diseases based on neural network.
5. Software implementation of the above algorithm and method.

The object of study is a set of images of wheat leaves.

In this work, we consider the essential methods for diagnosing plant diseases using images, as methods based on a neural network, and a method using texture analysis and fuzzy logic. On our data set, we performed the method of analysis of texture characteristics to select characteristics for classifying the type of disease. The classification is implemented using the fuzzy logic method and the fully connected network method, and then the results of the two methods are compared. Before calculating the texture features of the images, we used the classical image processing method using the OpenCV library to remove noise from the image — areas of

unnecessary information. Image processing helps to improve the quality of input into the system, regardless of what diagnostic method is used.

As a result of the work, a system for recognizing plant diseases from leaf images was developed using two methods: the fuzzy logic method and the neural network method. To compare the result of modeling the work of two methods, it turned out that with a small data set, the fuzzy logic method achieves an accuracy of more than 95%. And the neural network method is used only with a large data set for training. We tested the developed system on a wheat data set, but we also propose to apply it to other types of plants whose diseases are displayed on their leaves, for example, corn

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	10
1. АНАЛИЗ ИЗВЕСТНЫХ МЕТОДОВ ДИАГНОСТИКИ БОЛЕЗНЕЙ РАСТЕНИЙ ПО ИЗОБРАЖЕНИЙ ЛИСТЬЕВ	12
1.1 На основе нейронной сети.....	14
1.2 С использованием текстурного анализа и нечеткой логики	17
2. АЛГОРИТМ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ РАСПОЗНАВАНИЯ БОЛЕЗНЕЙ РАСТЕНИЙ ПО ИЗОБРАЖЕНИЯМ ЛИСТЬЕВ.....	19
2.1 Метод нормализации фотоизображений листьев	19
2.2 Алгоритм диагностики болезней растений.....	29
2.2.1 Матрица смежностей (GLCM), параметры Харалика....	30
2.2.2 Алгоритм на нечеткой логики.....	34
2.2.3 Эталонное описание и гистограммы	36
2.2.4 Алгоритм на нейронной сети	38
2.3 Реализация программной диагностики	41
2.3.1 Инструменты для разработки.....	41
2.3.2 Архитектура реализации	43
2.3.3 Тестирование	45
3. РЕЗУЛЬТАТ МОДЕЛИРОВАНИЯ РАЗРАБОТНОЙ СИСТЕМЫ РАСПОЗНАВАНИЯ.	48
3.1 Результат программы на нечеткой логики	49
3.2 Результат программы на нейронной сети.....	51

4. ЭКСПЕРИМЕНТАЛЬНЫЕ РЕЗУЛЬТАТЫ РАБОТЫ СИСТЕМЫ РАСПОЗНАВАНИЯ НА РЕАЛЬНЫХ ИЗОБРАЖЕНИЯХ ЛИСТЬЕВ ПШЕНИЦЫ.....	52
ЗАКЛЮЧЕНИЕ	57
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	60
ПРИЛОЖЕНИЕ 1: КЛАСС НОРМАЛИЗАЦИИ ИЗОБРАЖЕНИЯ	62
ПРИЛОЖЕНИЕ 2: КЛАСС ВЫЧИСЛЕНИЯ ПАРАМЕТРОВ ХАРАЛИКА.....	68
ПРИЛОЖЕНИЕ 3: КЛАСС ДИАГНОСТИКИ НА НЕЧЕТКОЙ ЛОГИКЕ.....	75
ПРИЛОЖЕНИЕ 4: ПРОГРАММА ДИАГНОСТИКИ НА НЕЙРОННОЙ СЕТИ	82

ВВЕДЕНИЕ

С древних времен в моей стране - Вьетнаме сельское хозяйство всегда играло важную роль в экономическом развитии страны. Вьетнам является одним из крупнейших в мире экспортеров риса, чая и кофе. Сегодня, благодаря сильному развитию эпохи технологий 4.0, высокотехнологичное сельское хозяйство, использующее достижения науки и техники, развивается стремительно, достигая звездных результатов. Благодаря научным и технологическим достижениям это помогает сократить трудозатраты, но при этом достичь таких же или даже лучших показателей.

Одной из важнейших задач при производстве сельскохозяйственной продукции является контроль качества продукции. Чтобы сельскохозяйственная продукция достигала хорошего качества, растения должны быть здоровыми и не иметь болезней. Поэтому фермерам необходимо регулярно отслеживать и проверять состояние заболевания растения, а также связываться с профессиональными людьми, врачами для лечения при обнаружении больного растения. Этот процесс занимает много времени, а также деньги фермеров. Это приводит к необходимости автоматизации этого процесса, помогая фермерам экономить время и производственные затраты, обеспечивая при этом своевременный мониторинг и обработку растений. Поэтому мы проводим исследования и разрабатываем диагностическую систему для растений, используя только листья. На больших полях система может быть интегрирована в беспилотное оборудование для мониторинга и проверки состояния заболеваний растений. Система практически осуществима, потому что большинство болезней растений отражаются в изменениях листьев, поэтому мы можем положиться на знаки на листьях для диагностики болезней растений.

Для решения задачи выделения особенностей на изображениях с целью их классификации (диагностики заболеваний) применяются различные методы формирования набора признаков, позволяющих однозначно идентифицировать изображения, т.е. относить их к определённому классу. Наиболее часто для решения задачи выделения особенностей на изображениях листьев растений с целью классификации вида заболевания растений используются методы нечёткой логики и нейронные сети, а диагностика производится как непосредственно по цветным RGB или HSV изображениям листьев, так и по их текстурным описаниям.

Наибольшее применение при решении задач распознавания болезней растений по изображениям листьев нашли признаки текстуры, использующие матрицы смежности (матрица GLCM для полутоновых изображений), признаки, основанные на измерении пространственных частот, признаки, использующие статистические характеристики изображений (среднее, энергия, вариация, однородность, контраст, коэффициент корреляции, энтропия, дифференциальная дисперсия), признаки, основанные на описании структурных элементов.

В данной работе, мы применяем такое применение и теорию нечёткой логики, чтобы распознавать болезни растений. В рамках этой работы объектом нашего исследования является пшеница - пища, которая чрезвычайно важна для человека.

1. АНАЛИЗ ИЗВЕСТНЫХ МЕТОДОВ ДИАГНОСТИКИ БОЛЕЗНЕЙ РАСТЕНИЙ ПО ИЗОБРАЖЕНИЙ ЛИСТЬЕВ

Задача диагностики болезней растений по изображениям листьев состоит из двух подзадач: обнаружения больных листьев растений в изображении, и классификации болезней листьев.

Обнаружение больных листьев растений в изображении может быть путем применением модели нейронной сети для обработки, сегментации изображения или использованием классических методов обработки изображений. В результате, больной лист растения выделяется из изображения, и требуется классификацию болезней этого листа [12].

Таблица 1 Известные методы классификации

Метод	Описание	Преимущества	Недостатки
K-Nearest Neighbouring	Рассчитайте минимальное расстояние между точками.	Легко реализовать и довольно хорошо в результате.	Медленное обучение, не устойчивый к шумовым данным в большом учебном примере.
Super Vector Machine (SVM)	Постройте гиперплоскость в бесконечномерном пространстве.	Точность прогноза высока, надежно работает, когда пример обучения содержит ошибки.	Привлекать длительное время обучения, трудно понять изученную функцию. Большой номер опорных векторов, используемых из учебного набора для выполнения задачи

			классификации.
Probabilistic Neural Network (PNN)	Он работает с четырехслойной структурой, которая включает в себя вычисление расстояния, хранение переменной предиктора и сравнение голосов.	Намного быстрее и точнее.	Требуется большое место для хранения.
Fuzzy Logic	Используйте функцию принадлежности для преобразования значения данных реального мира в степень принадлежности.	Высокая скорость, предпочтительнее при ограниченной точности I.	Размерность (большое количество черт).
Artificial Neural Network (ANN)	Многопользовательское восприятие - это базовая форма ANN, которая обновляет вес посредством базового распространения .	Хороший потенциал со способностью обнаруживать болезни листьев растений.	Требовать больше времени.

В таблице [12] описывается несколько популярных методов классификацию болезней. С своими преимуществами,

алгоритмы на основе нейронной сети и нечеткой логике более подходят нашей задаче и базе данных. Сейчас смотрим более конкретнее как в мире исследователи применяют этим алгоритмами для задачи диагностики болезней растений по изображениям листьев.

1.1 На основе нейронной сети

Подход нейронной сети (глубокого обучения Deep Learning - DL) - это подкатегория машинного обучения (Machine Learning - ML), введенная в 1943 году, когда была введена пороговая логика для построения компьютерной модели, очень похожей на биологические умы человека. Когда DL архитектуры начала развиваться с течением времени, исследователи применили их в задачи распознавания образов и классификации. Для оценки этих алгоритмов/архитектур использовались различные показатели производительности. Среди этих метрик: ошибка топ-1% / топ-5%, точность и отзыв, оценка F1, точность обучения/проверки и потери, но точность классификации (CA) являются наиболее популярными [10]. Для реализации моделей глубокого обучения требуется несколько шагов, от сбора наборов данных до отображений визуализации, которые описаны на рисунке 1.1.

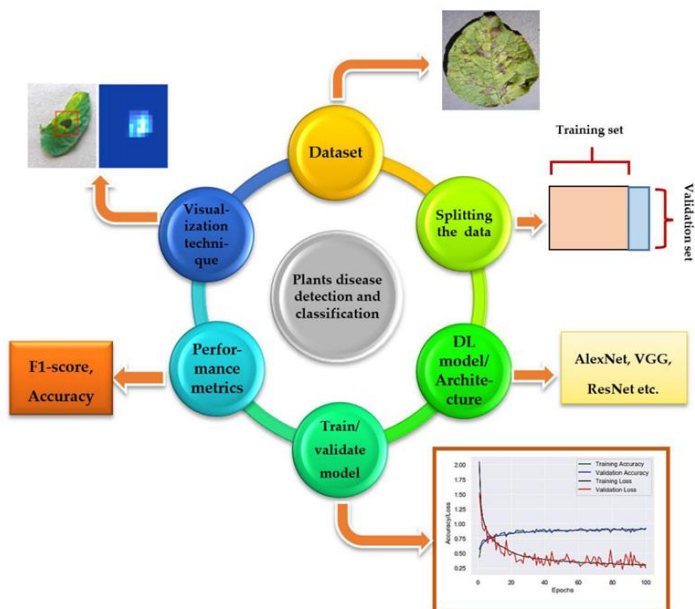


Рисунок 1. 1 Блок-схема реализации DL: сначала собирают набор данных, затем разделяют на две части, обычно на 80% обучения и 20% набора проверки. После этого модели DL обучаются с нуля или с использованием техники трансферного обучения, и их графики обучения / проверки получают, чтобы указать на значимость моделей. Затем показатели производительности используются для классификации изображений (тип конкретного заболевания растений), и, наконец, методы визуализации/отображения используются для обнаружения / локализации / классификации изображений.

Эти архитектуры также были реализованы для различных сельскохозяйственных приложений. Например, в [7] классификация листьев была выполнена с использованием модифицированного автором классификатора CNN и Random Forest (RF) среди 32 видов, у которых эффективность была оценена с помощью СА на уровне 97,3%. С другой стороны, он был не так эффективен при обнаружении закупоренных

объектов. Для классификации типов культур [8] использовался модифицированный автором CNN, [9] применялся VGG 16, [13] реализован три блока LSTM, а [14] использовался метод гистограмм CNN и RGB. [8] использовал CA, [9] использовал CA и пересечение над Union (IOU), [13] использовали CA и F1, и [14] использовали F1-счет в качестве метрики производительности. Среди них [14, 8] не было обеспечено точность обучения / валидации и потери. Кроме того, распознавание различных растений было сделано с помощью подхода DL. DL использовали модифицированный пользователем CNN, в то время использовали архитектуру AlexNet. Все были оценены на основе CA. DL превзошли два других с точки зрения CA. Точно так же дискриминация посевов / сорняков была проведена в [14], в которой автор предложил использовать CNN, и два набора данных были использованы для оценки модели. Оценил точность и отзыв; однако, получил CA для проверки предложенных моделей соответственно. Был изучен и достигнут уровень успеха 91.78% [16] идентификация растений со стороны DL подхода. Вдобавок ко всему, подходы DL также используются для критических задач, таких как выявление и классификация болезней растений, что является основной целью данного обзора. Существуют некоторые исследовательские работы, ранее представленные для подведения итогов исследований, основанных на сельском хозяйстве (включая распознавание болезней растений), выполненных DL, но в них отсутствуют некоторые недавние разработки с точки зрения методов визуализации, реализованных вместе с DL и модифицированных / каскадных. версия известных моделей DL, которые были использованы для идентификации болезней растений. Кроме того, этот обзор также предоставляет пробелы в исследованиях, чтобы получить более четкое / более прозрачное видение симптомов, наблюдаемых из-за болезней у растений.

1.2 С использованием текстурного анализа и нечеткой логики

Одним из самых популярных классических методов распознавания болезней растений по изображениям листьев является метод нечеткой логики использованием текстурного анализа изображения.

Классификаторы нечеткой логики - это системы классификации, которые используют нечеткие множества или нечеткую логику и преобразуют реальные значения данных в степени членства посредством использования функций принадлежности, так что эти правила затем можно использовать для процесса классификации. Это делается путем определения «категорий» для каждого из атрибутов.

Поскольку классификаторы нечеткой логики имеют очень высокую скорость, они предпочтительны в случаях, когда точность значений данных ограничена или, когда требуется классификация в режиме реального времени. Обработка нечетких изображений -это совокупность всех подходов, которые понимают, представляют и обрабатывают изображения, их сегменты и функции как нечеткие множества. Представление и обработка зависят от выбранного нечеткого метода и от решаемой проблемы. Обработка нечетких изображений делится на три основных этапа: фаззификация изображений, изменение значений принадлежности и, при необходимости, дефаззификация изображений. Из-за неопределенностей, которые существуют во многих аспектах обработки изображений, таких как аддитивный и неаддитивный шум при обработке изображений низкого уровня, неточности в предположениях, лежащих в основе алгоритмов, и неоднозначности в интерпретации во время обработки изображений высокого уровня, нечеткая обработка является желательной. Основным недостатком нечеткой

логики как классификатора является размерность, поскольку этот классификатор не подходит для задач, имеющих большое количество функций.

Текстурные характеристики большинство авторов рекомендуют находить, используя матрицы совместной встречаемости GLCM, которые наиболее точно отражают статистическую составляющую исследуемых изображений [11].

2. АЛГОРИТМ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ РАСПОЗНАВАНИЯ БОЛЕЗНЕЙ РАСТЕНИЙ ПО ИЗОБРАЖЕНИЯМ ЛИСТЬЕВ

2.1 Метод нормализации фотоизображений листьев

Выдалась выборка изображений для больных листьев пшеницы. Однако, исходные данные изображения больных листьев, в дополнение к изображениям больных листьев, также содержали много шума - области ненужной информации. Следовательно, для повышения точности системы диагностики, необходима реализация нормализации. Независимо от того, какой метод диагностики используется. Потому что нормализация помогает повысить качество ввода в систему. Нормализацию изображения, как один из важных методов обработки изображений, стоит рассматривать в виде преобразования исходного изображения в другое для улучшения интерпретируемости или восприятия информации людьми или обеспечения благоприятных условий для других методов автоматической обработки изображений [1].

Нормализация должна решить задачи :

- 1) Удаления неинформативных частей изображений (необходимо оставлять только изображения информативных частей листьев с признаками болезней);
- 2) Стандартизации формы, размеров и ориентации информативных частей листьев. Обработка много изображений показала, что оптимальный формат информативных частей изображений листьев пшеницы составляет 300*100 пикселей;

Алгоритм нормализации:

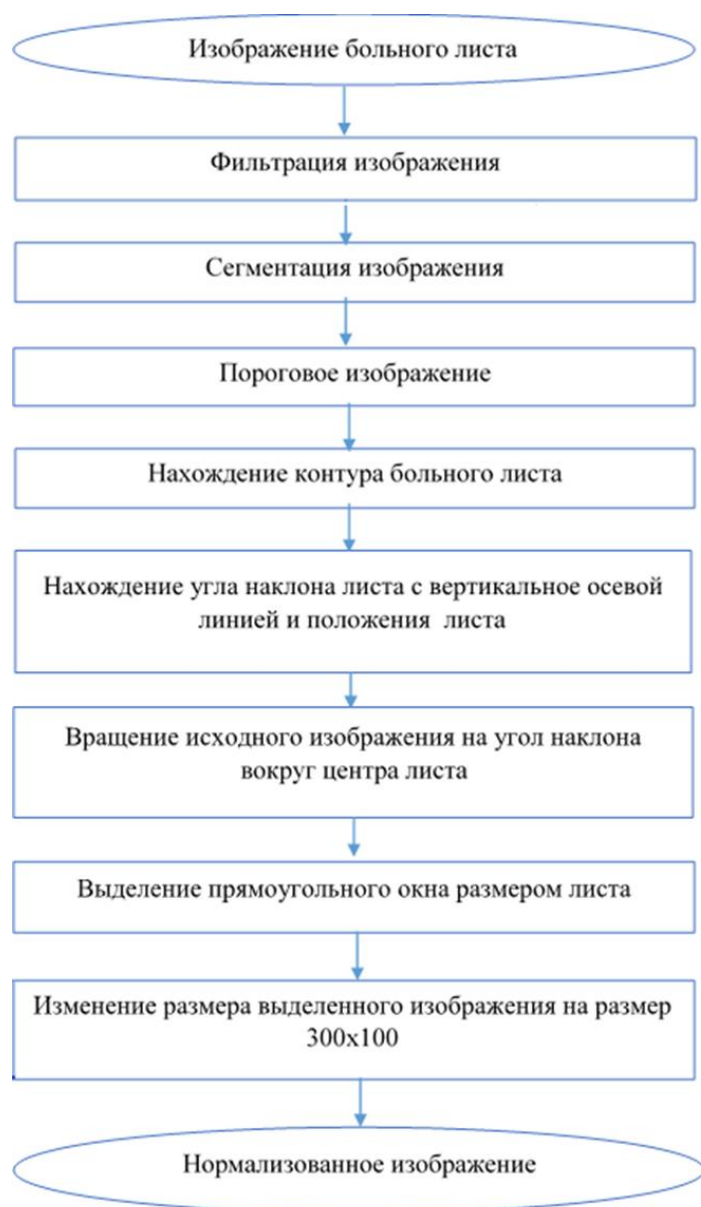


Рисунок 2. 1 Алгоритм нормализации

Сначала, на вход программы подает изображение больного листа пшеницы (рис 2.2). Как видно что, изображение содержит слишком много «шумов», которые могут влиять на результат вычисления характеристики и так же результат классификации. С этим изображением первое происходит – фильтрация.



Рисунок 2. 2 Исходное изображение больного листа пшеницы

Фильтрация изображения делается для того, чтобы зашумленное изображение привести к более «читаемом» виду. Безусловно, на сегодняшний день это не является обязательным шагом, однако неизвестно, насколько «смазанным» или «зашумленным» будет входное изображение, поэтому хорошим тоном считается фильтрация изображения.

В своей программе, я использовал 2 метода. Первый использовался метод – метод сдвига среднего значения [4].

Выход функции представляет собой отфильтрованное изображение с цветными градиентами и сплюсненной текстурой. В каждом пикселе (X, Y) входного изображения функция выполняет итерации со смещением, то есть рассматривается окрестность пикселя (X, Y):

$$(x, y): X - sp \leq x \leq X + sp;$$
$$Y - sp \leq y \leq Y + sp;$$

$$\|(R, G, B) - (r, g, b)\| < sr$$

Где sr – размер пространственного окна, а sr – размер цветового окна. (R, G, B) и (r, g, b) – векторы цветовых компонентов в (X, Y) и (x, y) соответственно. По окрестности находятся среднее пространственное значение (X', Y') и средний цветовой вектор (R', G', B') , и они действуют как центр соседства на следующей итерации. После завершения итераций цветовые компоненты исходного пикселя (то есть пикселя, с которого начались итерации) устанавливаются в конечное значение (средний цвет на последней итерации).



Рисунок 2. 3 Изображение после фильтрации

Далее, использовал оператор Лапласа. Дискретный оператор Лапласа часто используется в обработке изображений, например, в задаче выделения границ или в приложениях оценки движения. Дискретный лапласиан определяется как сумма вторых производных и вычисляется как сумма перепадов на соседях центрального пикселя. Метод усиления края по Лапласу рассматривает целый ряд различных ядер свертки. Наш ядер свертки:

$$kernel = \begin{matrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{matrix}$$

Как видно, сумма элементов матриц равна нулю, поэтому отклик фильтра может быть отрицательным. В этом случае значение отклика берется по модулю. В результате обработки области с постоянной или линейно возрастающей интенсивностью становятся черными, а области быстро изменяющихся значений интенсивности ярко высвечиваются.

После фильтрации, на изображении выполняется сегментация изображения алгоритмом по водоразделам Watershed.

Сегментация изображения — это разбиение изображения на множество покрывающих его областей. Сегментация применяется во многих областях, например, в производстве для индикации дефектов при сборке деталей, в медицине для первичной обработки снимков, также для составления карт местности по снимкам со спутников.

Алгоритм Watershed обработки изображений предназначен для отделения объектов на изображении от фона. Алгоритм принимает изображение в градациях серого и изображения маркера. Маркеры являются изображением, где вы указываете алгоритму Watershed объекты переднего плана и фон.

Алгоритм работает с изображением как с функцией от двух переменных

$$f = I(x, y)$$

где x, y – координаты пикселя. Значением функции может быть интенсивность или модуль градиента. Для наибольшего контраста можно взять градиент от изображения. Если по оси OZ откладывать абсолютное значение градиента, то в местах перепада интенсивности образуются хребты, а в однородных регионах – равнины. После нахождения минимумов функции f , идет процесс заполнения “водой”, который начинается с

глобального минимума. Как только уровень воды достигает значения очередного локального минимума, начинается его заполнение водой. Когда два региона начинают сливаться, строится перегородка, чтобы предотвратить объединение областей. Вода продолжит подниматься до тех пор, пока регионы не будут отделяться только искусственно построенными перегородками (рис 2.4).

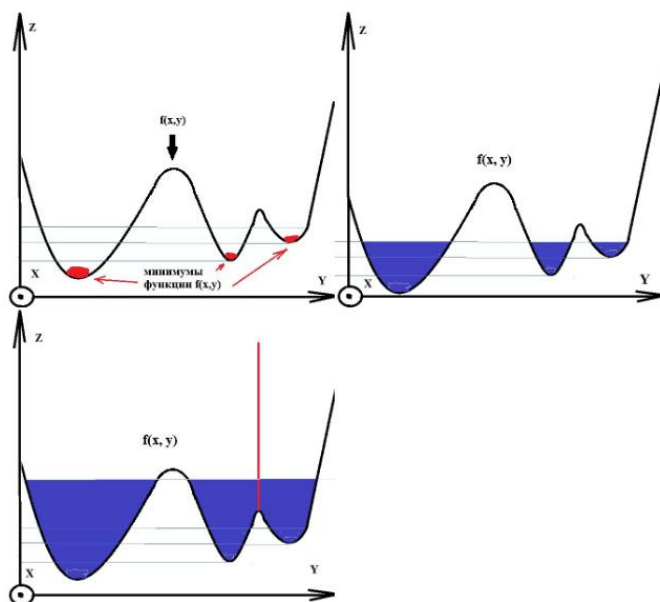


Рисунок 2. 4 Иллюстрация процесса заполнения водой

В результате работы алгоритма получается маска с сегментированным изображением, где пиксели одного сегмента помечены одинаковой меткой и образуют связную область. Основным недостатком данного алгоритма является использование процедуры предварительной обработки для картинок с большим количеством локальных минимумов (изображения со сложной текстурой и с обилием различных цветов).

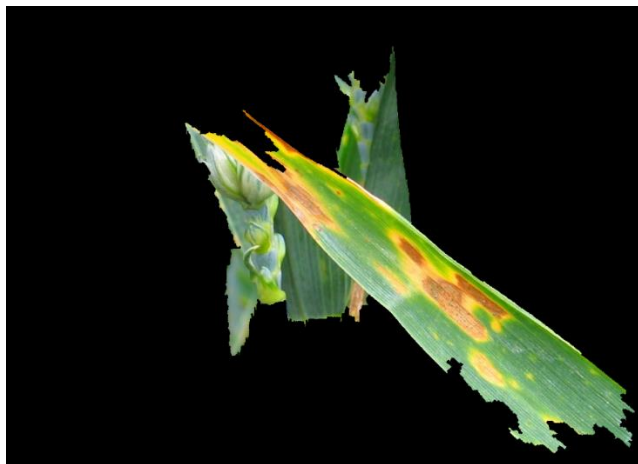


Рисунок 2. 5 Изображение после сегментации

После применения алгоритма Watershed segmentation, на изображение уже удалялось много неинформативных частей, осталось часть листа с болезнью и немного фонов. Чтобы выделить лист от фона, нужно найти края – контура листа. Использовал среднее пороговое значение. Здесь чтобы более гибко задать границы цвета, можно использоваться HSV формат.

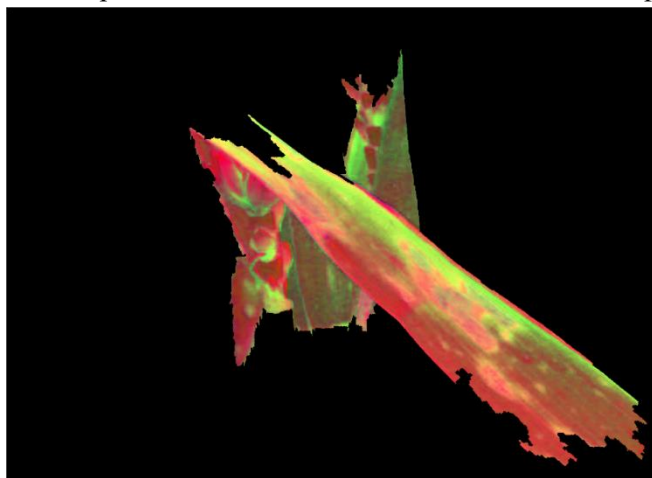


Рисунок 2. 6 Изображение в HSV формат

Алгоритм порога является бинаризацией Отцу (Otsu's Binarization). В глобальном пороговом значении мы использовали среднее значение для порогового значения. Итак, как мы можем знать, что выбранное нами значение является хорошим или нет? Ответ, метод проб и ошибок. Но рассмотрим бимодальное изображение (проще говоря, бимодальное изображение - это изображение, гистограмма которого имеет два пика). Для этого изображения мы можем приблизительно принять значение в середине этих пиков в качестве порогового значения, верно? Это то, что делает бинаризация Оцу. Простыми словами, он автоматически вычисляет пороговое значение по гистограмме изображения для бимодального изображения (Для изображений, которые не являются бимодальными, бинаризация не будет точной).

Для этого используется функция `threshold()`, но передается дополнительный флаг `CV_THRESH_OTSU`, кроме флага `CV_THRESH_BINARY`. Для порогового значения просто введите среднее значение. Затем алгоритм находит оптимальное пороговое значение. И пороговое значение Оцу используется в случае когда наше вводное пороговое значение не является хорошим. Поскольку мы работаем с бимодальными изображениями, алгоритм Оцу пытается найти пороговое значение (t), которое минимизирует взвешенную дисперсию внутри класса, определяемую соотношением:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

Где

$$q_1(t) = \sum_{i=1}^t P(i) \quad \& \quad q_2(t) = \sum_{i=t+1}^I P(i)$$

$$\mu_1(t) = \sum_{i=1}^t \frac{iP(i)}{q_1(t)} \quad \& \quad \mu_2(t) = \sum_{i=t+1}^I \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^t [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad \& \quad \sigma_2^2(t) = \sum_{i=t+1}^t [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

На самом деле он находит значение t , которое лежит между двумя пиками, так что отклонения от обоих классов минимальны.

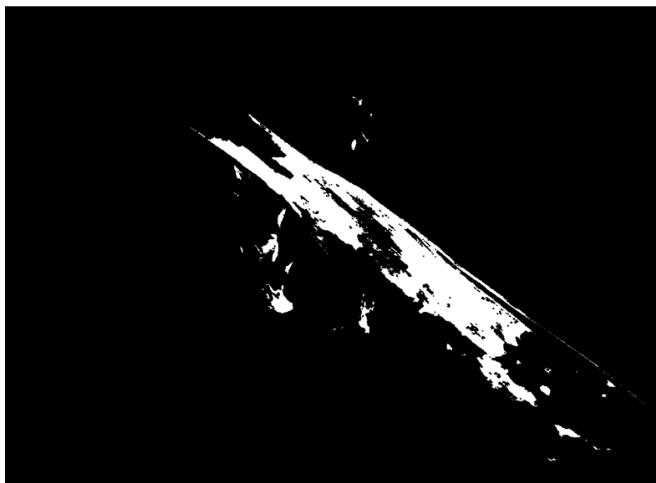


Рисунок 2. 7 Изображение после бинаризации

В результате пороговой операции, получается изображение на уровне серого, что облегчает последующие операции более высокого уровня, такие как обнаружение, идентификацию объектов или применение различного рода морфологических операций. С помощью морфологических операций MORPH_ELLIPSE, MORPH_CLOSE [4], я обнаружил контур листа, который имеет самую большую площадь. По найденному контуру, определяется положение листа и находится угол наклона листа с вертикальной осевой линией. Угол наклона является углом наклона эллипса, который покрывает контур листа, с вертикальной осевой линией.



Рисунок 2. 8 Отмеченное изображение

Далее, исходное изображение вращается на этот угол наклона вокруг центра листа. Из вращаемого изображения, выделяется прямоугольное окно размером главных осей этого эллипса. Потом размер выделенного окна изменяется на 300x100. 300x100 является стандартным размером, который мы используем дальше для вычисления характеристики нормализованного изображения.



Рисунок 2. 9 Вращенное изображение



Рисунок 2. 10 Нормализованное изображение

2.2 Алгоритм диагностики болезней растений

После процесса предобработки изображений, я получил набор нормализованных изображений больных листьев. Показано на рисунке 2.11.



Рисунок 2. 11 Примеры нормализованных изображений листьев пшеницы, поражённых болезнями. 1 – бурая ржавчина; 2 – темно-бурая пятнистость; 3 -мучнистая роса; 4 – пиренофороз; 5– септориоз; 6 – стеблевая головня; 7-полосатая мозаика; 8- желтая ржавчина

Как написано выше, чтобы классифицировать эти изображения по болезням, которые описаны в [2], можно применить метод нейронной сети. Типичный модель нейронной сети для классификации изображения является моделей CNN (Convolutional neural network). Как другие модели и методы нейронной сети, модель сверточной нейронной сети достигает ожидаемую точность только при условии таком что, набор данных имеет достаточный размер для обучения характеристики каждой болезни. К сожалению, у нас такой набор не хватает. Поэтому такой подход применения сверточной нейронной сети не является самым лучшим.

Обратите внимание, что на этом наборе данных, изображения различных заболеваний различаются друг с другом на основе текстурных характеристик изображения. С помощью этих характеристик можно классифицировать изображения на группы болезней. Кроме этого, текстурную характеристику можем увеличить быстро, легко и она так же содержит достаточную информацию изображения. Такая характеристика является характеристикой текстуры матрицы смежностей (Gray Level Co-Occurrence matrix), называемыми параметрами Харалика. Так как число характеристик (или признаков) может увеличиваться, поэтому мы можем классифицировать ее многими подходами, даже нейронной сети, потому что размер набора данных уже увеличивается намного.

2.2.1 Матрица смежностей (GLCM), параметры Харалика

Текстура - это свойство, которое представляет поверхность и структуру изображения, или ее можно определить как регулярное повторение элемента или рисунка на поверхности. Текстуры изображения - это сложные визуальные шаблоны, которые состоят из объектов или областей с под-шаблонами с характеристиками яркости, цвета, формы, размера и т. д.

Постоянная текстура в изображении представляет собой набор его характеристик, которые являются постоянными, постепенно изменяющийся или приблизительно периодический. Также это можно рассматривать как группировку сходства в изображении. Анализ текстуры характеризует пространственное изменение структуры изображения на основе некоторых математических процедур и моделей для извлечения информации из него. Анализ текстуры так же популярен в работах с медицинскими изображениями. Один из самых ранних методов, использованных для выделения текстурных признаков, был предложен Haralick Robert M. в 1973 году [15], известная как матрица кокуренса на сером уровне (матрица смежности - GLCM), и с тех пор она широко используется во многих приложениях для анализа текстур.

GLCM (gray-level co-occurrence matrix) – по своей сути это некая гистограмма значений градаций серого с заданным смещением по изображению, другими словами это полутонная матрица смежности.

Матрица GLCM создается путем вычисления частоты пикселя со значением интенсивности (уровня серого) i , возникающего в конкретной пространственной взаимосвязи с пикселем со значением j . Размер матрицы GLCM является количеством градаций уровней серого, может быть 4, 8, 16. Каждый элемент (i, j) в GLCM представляет собой просто общее количество раз, когда пиксель со значением i встречается в заданной пространственной зависимости с пикселем со значением j во входном изображении.

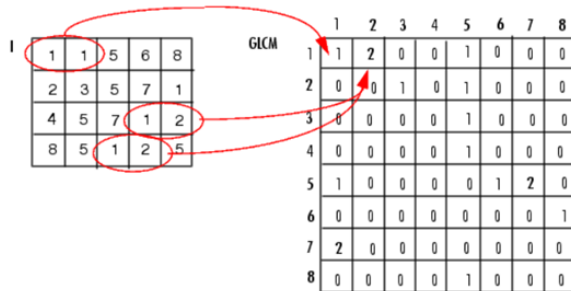


Рисунок 2. 12 Путь создания матрицы смежности

Пространственные зависимости могут быть вычислены для четырёх возможных направлений (рис 2.13) [18], и некоторых расстояний (D) между пикселем интереса и его соседнем, например 1, 2, 3, ... Видно что, из одного изображения, мы можем получить 4, 8, 12, ... матриц смежности. Плюс что, полное количество параметров Харалика – 14. Таким образом, максимальное количество текстурных параметров Харалика составит 56, 112, 168, 224, ... Таким образом, из одного изображения, мы имеем большие данные характеристик, которые содержат важную информацию изображения и могут быть исходными данными классификатора.

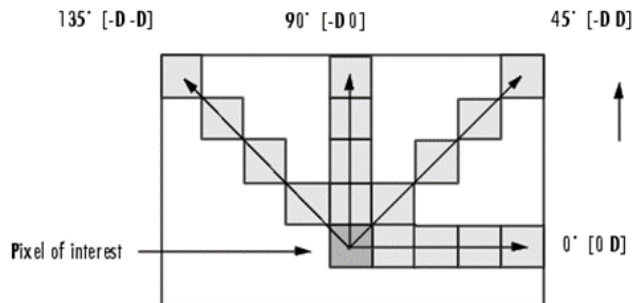


Рисунок 2. 13 Возможные направления пространственной зависимости

Количество параметров Харалика достаточно много, но не все они являются полезными, поэтому их применение зависит от

конкретных требований задачи.

Перед вычислением матрицы GLCM и подсчётом показателей, исходные нормализованные цветные изображения растений, представленные в цветовой модели RGB, были разделены на шесть цветовых компонентов: R(красная), G(зеленая), B(синяя), RG(разностная составляющая R и G), RB(R и B) и GB(G и B). На основе RGB – нормализованного изображения листа растения могут быть получены 6 GLCM матриц для компонент R, G, B, RG, RB, GB, для каждой из GLCM нормализованных матриц вычислены основные характеристики текстуры, называемыми параметрами Харалика: Contrast (Контрастность) – показатель степени локальных изменений в матрице GLCM, Correlation (Корреляция) - показатель вероятности совместного появления указанных пар пикселей, Energy (Энергия) - представляет сумму квадратов элементов в GLCM, также известен как равномерность или второй угловой момент, Homogeneity (Однородность) -степень близости распределения элементов в GLCM к диагонали GLCM [15, 18]:

1) contrast:

$$CN = \frac{1}{(G-1)^2} \sum_{u=0}^{G-1} \sum_{v=1}^{G-1} |u-v|^2 p(u, v)$$

2) correlation

$$CR = \frac{1}{2} \sum_{u=0}^{G-1} \sum_{v=0}^{G-1} \frac{(u - \mu_u)(v - \mu_v)}{\sigma_u^2 \sigma_v^2} p(u, v) + 1$$

3) energy

$$EN = \sum_{u=0}^{G-1} \sum_{v=0}^{G-1} p(u, v)^2$$

4) homogeneity

$$HM = \sum_{u=0}^{G-1} \frac{p(u, v)}{1 - |u - v|}$$

где u, v - координаты матрицы смежности, G - количество градаций уровней серого, μ_u, μ_v, σ_u и σ_v - средние значения и стандартные отклонения u -й строки и v -го столбца матрицы совпадения соответственно. Приведённые выше определения гарантируют, что все функции имеют диапазон $[0, 1]$.

2.2.2 Алгоритм на нечеткой логики

Прямое использование параметров Харалика для определения типа заболевания не приводит к однозначно правильному результату распознавания. Для генерации однозначных результатов распознавания применяется нечеткая логика.

Возможность его использования при решении задачи диагностики заболеваний растений по изображениям листьев рассмотрена в [3]. Отличительной особенностью нашего решения этой проблемы является то, что она включает в себя вычисление функций принадлежности для каждого из 6 наборов R, G, B, RG, RB, GB , бинаризацию результатов и принятие окончательного решения об изображении, принадлежащем одному из N возможных типов большинством голосов.

Обучение заключается в получении эталонного описания в виде матрицы значений среднеквадратических отклонений, математических ожиданий параметров Харалика для каждой цветовой компоненты R, G, B, RG, RB, GB исходных RGB фотоизображений для всех N болезней (табл. 2а).

$k+j*4$						
i	1	2	3	4	N
0	A_{11}	A_{12}	A_{13}	A_{14}	A_{1n}
1	A_{21}	A_{22}	A_{23}	A_{24}	A_{2n}
2	A_{31}	A_{32}	A_{33}	A_{34}	A_{3n}
3	A_{41}	A_{42}	A_{43}	A_{44}	A_{4n}
4	A_{51}	A_{52}	A_{53}	A_{54}	A_{5n}
5	A_{61}	A_{62}	A_{63}	A_{64}	A_{6n}

$k+j*4$	
0	B_1
1	B_2
2	B_3
3	B_4
4	B_5
5	B_6

6	A ₇₁	A ₇₂	A ₇₃	A ₇₄	A _{7n}
7	A ₈₁	A ₈₂	A ₈₃	A ₈₄	A _{8n}
8	A ₉₁	A ₉₂	A ₉₃	A ₉₄	A _{9n}
...
m	A _{m1}	A _{m2}	A _{m3}	A _{m4}		A _{mn}

6	B ₇
7	B ₈
8	B ₉
...	...
m	B _m

Таблица 2 а) Эталонное описание б) Вектор параметров исходного изображения

Здесь i – номер болезни, k – номер параметра Харалика для каждой цветовой компоненты, j – номер цветового компонента, например k=0 будет соответствовать цветовой компоненте – R, параметру – Contrast, k=1 – цветовой компоненте – R, параметру – Correlation и т. д. Значение N = 8, k=0..3, j = 0..5. Все параметры нумеруются от 0 до m = 23.

Формат набора усреднённых значений параметров Харалика диагностируемых фотоизображений представляет вектор-столбец (табл. 2б).

Основные этапы алгоритма [6]:

- 1) вычисление функции принадлежности (membership function):
(показатели сравнения основных параметров исходного изображения с эталонными описаниями) для всех компонентов и всех заболеваний:

$$MF(i, k, j) = \frac{\sum_{j=0}^5 \sum_{k=0}^3 (A(j * 4 + k, i) - \overline{A(j * 4 + k, i)})(B(j * 4 + k) - \overline{B(j * 4 + k)})}{\sqrt{\sum_{j=0}^5 \sum_{k=0}^3 (A(j * 4 + k, i) - \overline{A(j * 4 + k, i)})^2 \sum_{j=0}^5 \sum_{k=0}^3 (B(j * 4 + k) - \overline{B(j * 4 + k)})^2}}$$

Где:

A(j*4+k, i) – элемент столбца i, строки (j*4+k) матрицы эталонного описания;

B(j*4+k) – (j*4+k)-тый элемент вектора – столбца

усреднённых параметров Харалика диагностируемых образцов листьев растений;

- 2) Первичная бинаризация показателей на пороге P:

$$BMF(i, k, j) = 1 \text{ если } MF(i, k, j) \geq P \text{ или } 0 \text{ если } MF(i, k, j) < P$$

- 3) Суммирование бинаризованных показателей и вторичной бинаризации:

$$SBMF(k, j) = \sum_{k=1}^4 BMF(i, k, j)$$

$$B2MF(i, j) = 1 \text{ если } SBF(k, j) = \max(SBF(k, j)) \text{ или } 0$$

- 4) Суммирование результатов вторичного бинаризации, итогового большинства голосов и бинаризации:

$$BG(i) = \sum_{j=1}^6 B2MF(i, j)$$

$$K(i) = 1 \text{ если } BG(i) = \max(BG(i)) \text{ или } 0$$

По итоговым голосам, система определена тип заболевания исходных параметров. Тип заболевания соответствует индексу массива K, где значение элемента равно 1.

2.2.3 Эталонное описание и гистограммы

Для реализации алгоритма на нечеткой логике, нужно построить эталонное описание. И как написано выше, эталонное описание описывается в виде матрицы значений среднеквадратических отклонений, математических ожиданий параметров Харалика для каждой цветовой компоненты R, G, B, RG, RB, GB исходных RGB фотоизображений для всех N болезней (табл. 2). Поэтому для построения его следует анализировать распределение параметров Харалика чтобы вычислить значения среднеквадратических отклонений и математических ожиданий.

Все значения параметров Харалика для каждой цветовой компоненты для всех болезней находятся в промежутке $[0, 1]$. После чего, мы можем построить гистограммы, представляющие собой распределение плотности вероятности из элементов векторов параметров (признаков).

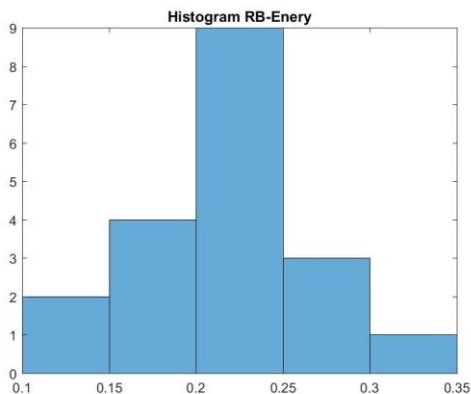


Рисунок 2. 14 Гистограмма распределения параметров Energy на компоненте RB

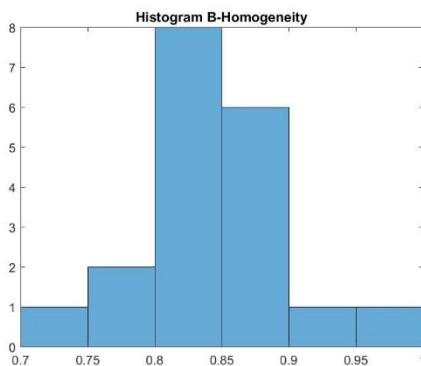


Рисунок 2. 15 Гистограмма распределения параметров Homogeneity на компоненте B

Однако, основываясь исходя из центральной предельной теоремы, какова бы ни была форма гистограммы, выборочное распределение всегда будет стремиться к

нормальному [17]. Это значит, что математическое ожидание будет совпадать с центром распределения. Именно математическое ожидание будет отражать статистическую особенность для определенного вида болезни, в виду того, что в вычислении математического ожидания задействованы все элементы выборки.

Теперь необходимо и достаточно лишь найти максимальную точку кривой аппроксимации, после чего вычислить центр распределения. Дополнительными характеристиками распределений плотности также являются среднеквадратическое отклонение и диапазон доверительного интервала.

Хотя наше выборочное распределение не действительно стремится к нормальному, но по правилу Бьенеме-Чебышева, мы можем убедить что большие наши параметры находятся в доверительном интервале, который почти совпадает с доверительным интервалом нормального распределения. Бьенеме и Чебышева обнаружили, что для любого набора данных, независимо от формы распределения, процент наблюдений, лежащих на расстоянии не превышающем k стандартных отклонений от математического ожидания, не меньше $(1 - 1/k^2) * 100\%$.

Вот так, по этому подходу, мы получили эталонное описание для реализации алгоритма диагностики основанного на теории нечеткой логики. Полученное эталонное описание сохраняется в отдельном файле вида csv (comma-separated values) чтобы каждый раз запуски программу диагностики не требуется вычисление его снова.

2.2.4 Алгоритм на нейронной сети

В настоящее время, алгоритмы нейронной сети все чаще применяются и также доказали свою эффективность в задачах классификации, идентификации, ... Для моей задачи - иметь

возможность классифицировать болезни растений на основе характеристик Харалика, рассчитанных по приведенной выше формуле для каждого изображения, также можно использовать метод нейронной сети. Но, как упоминалось выше, каждая модель нейронной сети работает только тогда, когда она снабжена набором данных, достаточно большим для обучения. Поэтому в рамках этой работе, я создал модель нейронной сети для диагностики заболеваний растений по параметров Харалика, так как количество параметров Харалика намного больше чем количество имеющих изображений, и так же может увеличиваться легко. И анализа минимального размера набора данных, необходимого для достижения хорошей работы модели.

Выбранный модель является полностью подключенной нейронной сети (Fully connected neural network – FNN) [5]. Этот модель чаще добавляется в модель классификации изображения, построенный по модели CNN (Convolutional neural network). Он добавляется после слоев, которые выполняются сверточные операции изображения. Сверточные слои играет в роль выделения характеристики изображения. Поэтому на самом деле, часть FNN принимает на входе характеристики изображения.

Полностью подключенная нейронная сеть состоит из серии полностью связанных слоев [5]. Полностью подключенный слой - это функция от R^m до R^n . Каждое выходное измерение зависит от каждого входного измерения. Наглядно, полностью связанный слой представлен следующим образом на рисунке

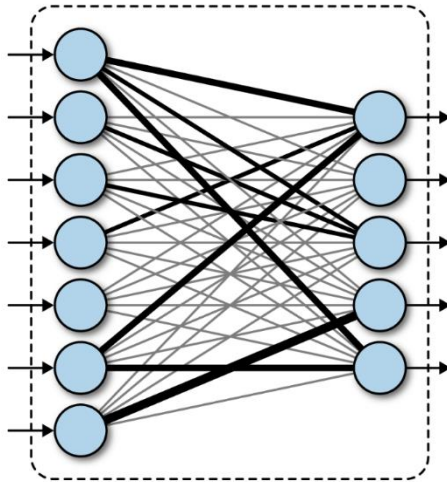


Figure 4-1. A fully connected layer in a deep network

Рисунок 2. 16 Полносвязные слои в глубокой сети
Структура нейронной сети:

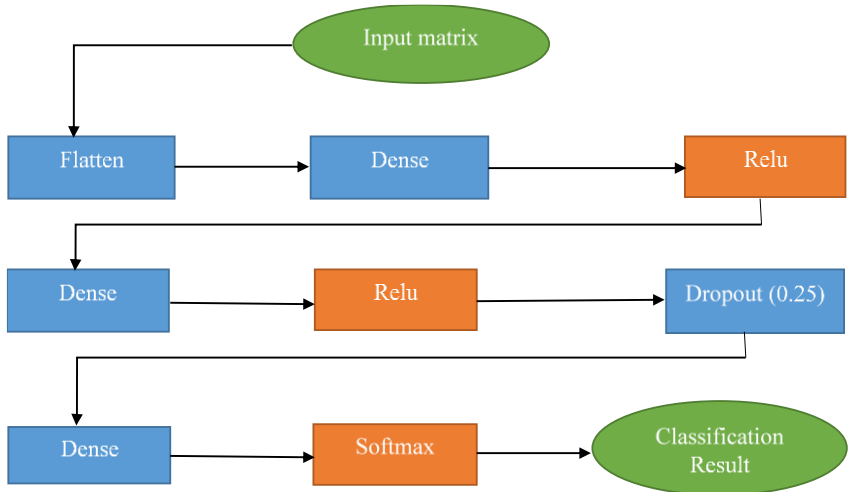


Рисунок 2. 17 Структура модели FNN классификации изображений по параметрам

На входе сети, Input matrix является матрицей эталонного описания. Перед тем, как данные попадет в слой Dense, нужно

через слой Flatten чтобы преобразовать матрицу в вид вектора потому что слой Dense принимает на входе данные вида вектора.

Кроме этого, в сети добавлен одной слой Dropout с коэффициентом 0.25. Это слой используется для избежания проблемы переобучения в нейронной сети.

Так как, наша задача является однозначной многоклассовой классификацией, поэтому на последнем слое, я использовал функцию активации Softmax чтобы вычислять вероятности принадлежности одной входной матрицы к одному классу заболевания.

2.3 Реализация программной диагностики

2.3.1 Инструменты для разработки

Для того чтобы моделировать и демонстрировать работу системы распознавания болезней растений, была выбрана операционная система Windows 10, и среда Visual Studio 2017 Community. Система написана на языке C++, используя открытую библиотеку OpenCV в целях повышения производительности.

OpenCV (Open Source Computer Vision Library) - это библиотека программного обеспечения для компьютерного зрения и машинного обучения с открытым исходным кодом. OpenCV был создан для обеспечения общей инфраструктуры для приложений компьютерного зрения и для ускорения использования машинного восприятия в коммерческих продуктах. OpenCV позволяет предприятиям легко использовать и модифицировать код. Библиотека имеет более 2500 оптимизированных алгоритмов, которые включают полный набор как классических, так и самых современных алгоритмов компьютерного зрения и машинного обучения. Эти алгоритмы могут использоваться для обнаружения и распознавания лиц, идентификации объектов, классификации действий человека в

видео, отслеживания движений камеры, отслеживания движущихся объектов, извлечения трехмерных моделей объектов, создания трехмерных облаков точек из стереокамер, объединения изображений для получения высокого разрешения. изображение всей сцены, поиск похожих изображений из базы данных изображений, удаление красных глаз с изображений, снятых с использованием вспышки, отслеживание движений глаз, распознавание пейзажей и установка маркеров для наложения их на дополненную реальность и т. д. В OpenCV работает более 47 тысяч пользователей сообщество и предполагаемое количество загрузок, превышающее 18 миллионов. Библиотека широко используется в компаниях, исследовательских группах и государственных органах. Он имеет интерфейсы C++, Python, Java и MATLAB и поддерживает Windows, Linux, Android и Mac OS. OpenCV в основном ориентирован на приложения для визуализации в реальном времени и использует инструкции MMX и SSE, когда они доступны. Полнофункциональные интерфейсы CUDA и OpenCL сейчас активно развиваются. Существует более 500 алгоритмов и примерно в 10 раз больше функций, которые составляют или поддерживают эти алгоритмы.

Чтобы оценить результат работы системы, и так же сравнивать результат методов, которые основан на теории нечеткой логики, и на нейронной сети, была переписана программа классификации основанная на нейронной сети на язык Python 3, используя библиотеку Tensorflow – одна из самых популярных библиотек для машинного обучения в мире. На Python 3 так же поддерживается библиотеки OpenCV для обработки изображения.

2.3.2 Архитектура реализации

Программа основанная на нечеткой логике реализирует на языке C++ с помощью библиотеки OpenCV.

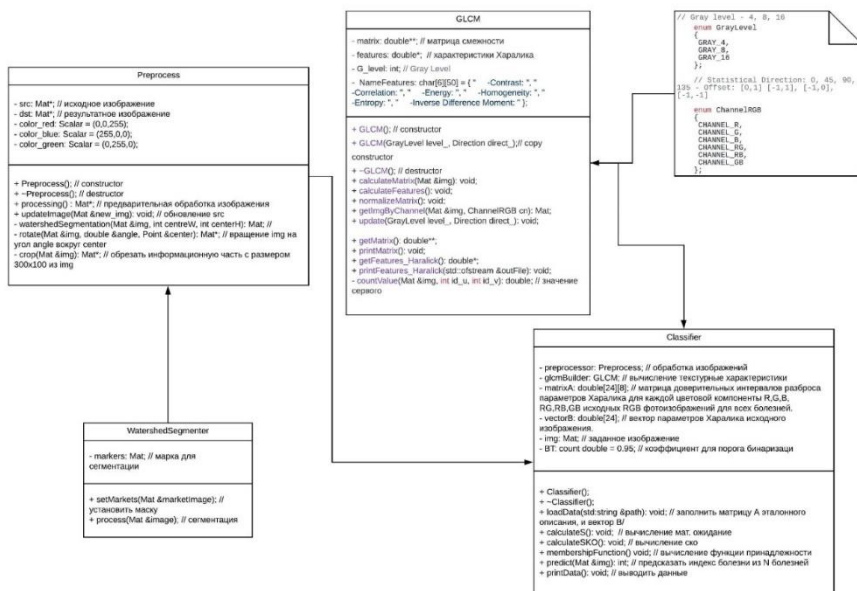


Рисунок 2. 18 Диаграмма классов

Класс WatershedSegmenter: реализация метода сегментации изображения алгоритмом Watershed. Использовать для загрузки маски и выполнения алгоритма.

Класс Preprocess: реализация метода нормализации фотоизображений листьев. На входе его попадает фотоизображение листьев. После выполнения функции processing(), на выходе получаем информационную часть изображения с размером 300x100 – нормализованное изображение.

Класс GLCM: вычисление векторов параметров Харалика для изображения. На входе его попадает нормализованное изображение, выполняется вычисление матрицы смежности

GLCM и текстурные характеристики Харалика. На выходе получаем вектор, содержащий характеристики GLCM: contrast, correlation, energy, homogeneity, entropy и inverse difference moment.

Класс Classifier: классификатор. В него, загрузить матрицу эталонного описания 8 болезней и вектор параметров Харалика тестированного изображения, и выполняется реализация алгоритма диагностики болезней растений. Здесь алгоритм диагностики болезней растений реализован на основе нечеткой логики.

Реализация алгоритма диагностики на основе нейронной сети выполняется на языке Python3, так как Python 3 более популярно легко для построения модели нейронной сети. Код определения модели FCN:

```
model = Sequential()  
model.add(Flatten(input_shape=(6,4)))  
model.add(Dense(24, activation='relu'))  
model.add(Dense(64, activation='relu'))  
model.add(Dropout(0.25))  
model.add(Dense(8, activation='softmax'))
```

Данные маленькие, поэтому модель так же небольшая. При этом подобрать параметры для того, чтобы нейронная сеть корректно работала, подобрать руками легко.

При этом в качестве оптимизатора был выбран Adam, в качестве функции потерь используется categorical_crossentropy. Перекрестная энтропия - это функция потерь, используемая для измерения различия между распределением наблюдаемых меток классов и предсказанными вероятностями принадлежности к классам. Категориальный относится к возможности иметь более двух классов (вместо двоичного, который относится к двум классам). Код, определяющий параметры компиляции модели:

```
model.compile(loss='categorical_crossentropy',
```

optimizer='adam', metrics=['accuracy'])

Метрика accuracy показывает долю правильных ответов алгоритма.

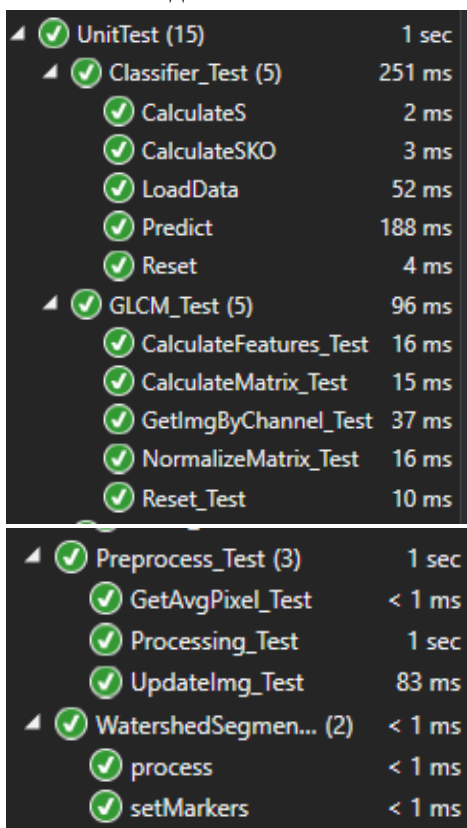
2.3.3 Тестирование

В процессе тестирования, мы проверили работы каждого отдельного модуля исходного кода программы.

Наш проект написан в Visual Studio, поэтому мы используем Testing tools in Visual Studio. UnitTestTool включают в себя:

- Test Explorer - запуск модульных тестов и просмотр их результатов в Test Explorer. Вы можете использовать любой фреймворк для модульного тестирования, в том числе сторонний фреймворк, который имеет адаптер для Test Explorer.
- Microsoft unit test framework for managed code - Среда модульного тестирования Microsoft для управляемого кода устанавливается вместе с Visual Studio и предоставляет платформу для тестирования кода .NET.
- Microsoft unit test framework for C++ - Среда модульного тестирования Microsoft для C++ устанавливается как часть разработки рабочего стола с рабочей нагрузкой C++. Он предоставляет основу для тестирования нативного кода. Платформы Google Test, Boost Test и CTest также включены, а сторонние адаптеры доступны для дополнительных платформ тестирования.
- Code coverage tools - Вы можете определить объем кода продукта, который выполняет ваше модульное тестирование, с помощью одной команды в Test Explorer.
- Microsoft Fakes isolation framework - Каркас изоляции Microsoft Fakes может создавать замещающие классы и методы для производственного и системного кода, которые создают зависимости в тестируемом коде.

Реализуя ложные делегаты для функции, вы управляете поведением и выводом объекта зависимости.



✔ UnitTest (15)	1 sec
✔ Classifier_Test (5)	251 ms
✔ CalculateS	2 ms
✔ CalculateSKO	3 ms
✔ LoadData	52 ms
✔ Predict	188 ms
✔ Reset	4 ms
✔ GLCM_Test (5)	96 ms
✔ CalculateFeatures_Test	16 ms
✔ CalculateMatrix_Test	15 ms
✔ GetImgByChannel_Test	37 ms
✔ NormalizeMatrix_Test	16 ms
✔ Reset_Test	10 ms
✔ Preprocess_Test (3)	1 sec
✔ GetAvgPixel_Test	< 1 ms
✔ Processing_Test	1 sec
✔ UpdateImg_Test	83 ms
✔ WatershedSegmen... (2)	< 1 ms
✔ process	< 1 ms
✔ setMarkers	< 1 ms

Рисунок 2. 19 Модульные тестирования

Все тесты, разделены по модулям (классам) соответственно с диаграммой классов, прошли с успешным результатом.

Как написано выше, в Visual Studio поддерживается Testing tool, который включает в себя: Code coverage tools. С помощью этого инструмента, мы подсчитала проценты покрытия кода.

Coverage					
Filter:	<input type="text"/>	<input checked="" type="checkbox"/> Display coverage			
	Coverage	Covered line	Uncovered line	Total line	
SystemDiagnostic.exe	92%	594	50	644	
I:\GradEs\Program\System\x64\Debug\SystemDiagnostic.exe	92%	594	50	644	
i:\grades\program\system\normimage\preprocess.h	88%	14	2	16	
i:\grades\program\system\normimage\classifier.cpp	88%	189	26	215	
i:\grades\program\system\normimage\glcm.cpp	91%	172	17	189	
i:\grades\program\system\normimage\preprocess.cpp	97%	134	4	138	
i:\grades\program\system\normimage\main.cpp	98%	61	1	62	
i:\grades\program\system\normimage\classifier.h	100%	22	0	22	
i:\grades\program\system\normimage\glcm.h	100%	2	0	2	

Рисунок 2. 20 Покрытие строк.

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
{ } Global Classes	65	5.19 %	1188	94.81 %
Classifier	38	7.85 %	446	92.15 %
GLCM	14	5.34 %	248	94.66 %
Global Functions	2	2.82 %	69	97.18 %
Preprocess	11	2.72 %	393	97.28 %
WatershedSegmenter	0	0.00 %	32	100.00 %

Рисунок 2. 21 Покрытие ветвей.

По строкам, процесс тестирования покрыт более 92% строк, а по ветвям, также более 94% покрыт. Результат тестирования доказывает, что, разработанная система удовлетворена, может применяться на работу. Почти все тест-кейсы уже проверили и получили успешные результаты.

3. РЕЗУЛЬТАТ МОДЕЛИРОВАНИЯ РАЗРАБОТНОЙ СИСТЕМЫ РАСПОЗНАВАНИЯ.

Так как выборка данных имеет не большой размер, поэтому чтобы оценить и так же сравнивать точности двух разработанные систем, сначала мы генерируем набор параметров Харалика. Сказалось выше, распределение значений параметров Харалика стремится к нормальному распределению. Поэтому мы генерируем набор по следующему алгоритму:



Рисунок 3. 1 Алгоритм генерирования параметров Харалика

Алгоритм реализован в среде Visual Studio на языке C++, в которое более легче выполняются вычисления математических ожиданий и генерирование нормальное распределение. Значение СКО выбирается чтобы по выбранному значению, точность распознавания для всех болезней на генерированный набор более 95%. То есть, для каждого значения СКО, мы генерируем 1000

тестовых значений параметров Харалика и проверяем результат распознавания программы нечеткой логики, если точность всех болезней более 95%, то это значение является хорошим. Мы выбираем максимальное значение СКО, которое удовлетворяется этому условию. Мы используем программу нечеткой логики потому что, эта программа на этапе обучения, используется только матрицу математических ожиданий, поэтому генерированные значения не влияют на этап обучения. Результат распознавания генерированных значений будет точнее. И в результате, получается 0.06 – максимальное значение СКО, при котором, точность на генерированный набор для всех болезней больше 95%.

После получения СКО, мы генерируем по каждой болезни 1000 векторов 24 значений параметров Харалика для обучения, и 40 векторов значений для проверки для каждой болезни.

3.1 Результат программы на нечеткой логики

Как написано, результат моделирования программой основанной на нечеткой логике, является генерированным набором данных, используемым для оценки работы системы диагностики.

На генерированный набор данных, мы анализируем объем данных, требуемый для обучения метода нечеткой логики.

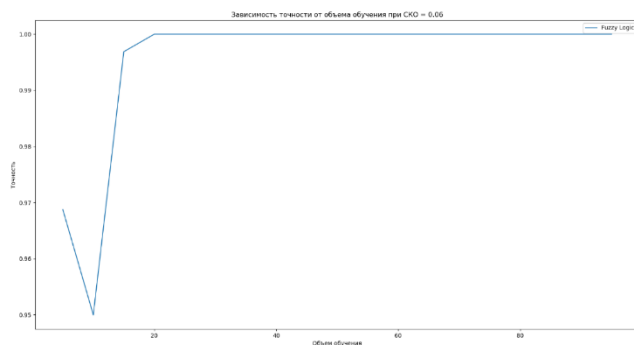


Рисунок 3. 2 Зависимость точности метода нечеткой логики от объема данных

На графике, видно что, метод нечеткой логики работает хорошо даже при малом наборе входных данных. С объема 15 векторов параметров Харалика для одной болезни, система уже достигается лучшую точность.

Полученный результат доказывает преимущество метод нечеткой логики для задачи с малым объемом данных на входе обучения. Такой хороший результат мы получили потому что метод нечеткой логики совершенно не сильно зависит от объема обучения. Суть метода основан на рассмотр распределения входных параметров, а распределение входных параметров, как известно заранее, стремится к нормальному, поэтому при малом объемом параметров, мы уже можем определить их распределение, математическое ожидание распределения. На графике тоже показывается это, когда объем входных данных увеличивается, точность устойчива, мало изменяется. Потому что, точное значение математических ожиданий уже получилось, когда объем данных достаточен, поэтому после достижения точного значения математических ожиданий, увеличение не имеет в роль увеличения точности, даже делает система медленнее работает. Потому что большие данные следуют

больше вычислений, и система работает медленнее.

3.2 Результат программы на нейронной сети

Программа нейронной сети определяется нужным объемом данных для обучения. На входе его так же набор генерированных данных, содержал по каждой болезни 1000 векторов параметров для обучения и 40 векторов параметров для тестирования.

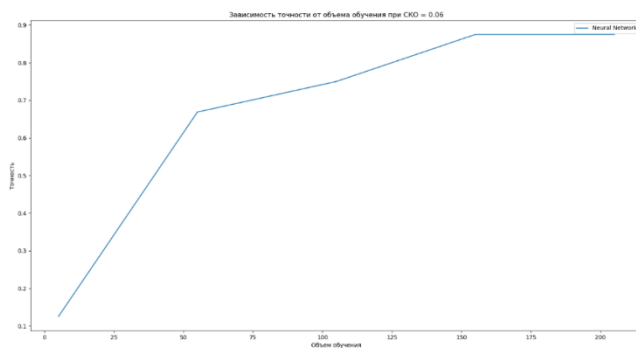


Рисунок 3. 3 Зависимость точности метода нейронной сети от объема данных

На графике показывается зависимость точности программы от объема данных. Результат доказывает, что точность метода нейронной сети сильно зависит от объема входных данных. Система диагностики работает хорошо только если на входе попадает набор с достаточным объемом данных.

В результате моделирования, мы получаем вывод о развитии системы диагностики болезней растений по изображениям листьев. Система может работать по следующим принципам: в начале, когда система еще не имеет большого объема данных, мы можем одновременно применять нечеткую логику для диагностики, и собираем данные, потом когда данные собираются достаточно, мы можем применять нейронную сеть, чтобы уменьшить вычислений внутри системы.

4. ЭКСПЕРИМЕНТАЛЬНЫЕ РЕЗУЛЬТАТЫ РАБОТЫ СИСТЕМЫ РАСПОЗНАВАНИЯ НА РЕАЛЬНЫХ ИЗОБРАЖЕНИЯХ ЛИСТЬЕВ ПШЕНИЦЫ

Как доказывалось выше, чтобы применить программу, основанную на нейронной сети, для распознавания реальное изображение листа пшеницы, требуется на этапе обучения как минимум 150 изображений каждой болезни. К сожалению, наша выборка данных не хватает. Но выше так же написано, что, программа, основана на нечеткую логику, работает хорошо с набором данных объема 10 изображений каждой болезни. Поэтому, наш эксперимент принимается программу нечеткой логики. Наша выборка данных разделяется на две подгруппы: обучения и проверки с отношением 5:1.

Таблица 3 Эталонное описание на реальных изображениях листьев пшеницы

ЭТАЛОННОЕ ОПИСАНИЕ

БОЛЕЗНЬ		1	2	3	4	5	6	7	8
R	Contrast	0,00739	0,00417	0,00661	0,00309	0,00555	0,00536	0,00599	0,00508
	Correlation	0,87628	0,96073	0,95176	0,97766	0,93771	0,96003	0,97406	0,94052
	Energy	0,19492	0,27936	0,11942	0,21614	0,25034	0,17677	0,39972	0,18587
	Homogeneity	0,84122	0,90743	0,85208	0,94044	0,89455	0,88314	0,91916	0,88228
G	Contrast	0,00746	0,00432	0,00568	0,00318	0,00575	0,00533	0,00643	0,00468
	Correlation	0,89082	0,97432	0,94264	0,96879	0,92916	0,96679	0,97187	0,93673
	Energy	0,17864	0,24277	0,15560	0,27778	0,22141	0,17194	0,33099	0,19877
	Homogeneity	0,83801	0,90623	0,86968	0,94002	0,89019	0,88460	0,91235	0,88899
B	Contrast	0,00701	0,00356	0,00600	0,00269	0,00545	0,00517	0,00828	0,00442
	Correlation	0,86002	0,95740	0,92987	0,95286	0,90284	0,95608	0,96712	0,92943
	Energy	0,21068	0,31602	0,16522	0,31838	0,25409	0,19425	0,10920	0,22209
	Homogeneity	0,84299	0,91865	0,86295	0,94379	0,89144	0,88644	0,86078	0,89469
RG	Contrast	0,00257	0,00023	0,00000	0,00034	0,00116	0,00000	0,00035	0,00144
	Correlation	0,91020	1,00000	1,00000	1,00000	0,93616	1,00000	1,00000	0,87185
	Energy	0,44906	0,94027	1,00000	0,80883	0,54295	0,99996	0,86985	0,57736
	Homogeneity	0,93866	0,99445	1,00000	0,99170	0,97163	0,99999	0,99138	0,96474
RB	Contrast	0,00453	0,00115	0,00228	0,00187	0,00230	0,00130	0,00476	0,00372

GB	Correlation	0,92505	0,92371	0,90748	0,97504	0,94662	0,92685	0,95813	0,93930
	Energy	0,21302	0,57111	0,45378	0,27131	0,36294	0,54174	0,17117	0,24136
	Homogeneity	0,89133	0,97196	0,94419	0,95640	0,94662	0,96816	0,90231	0,91136
	Contrast	0,00299	0,00115	0,00275	0,00167	0,00213	0,00130	0,00441	0,00230
	Correlation	0,84671	0,96388	0,89198	0,96257	0,91710	0,94254	0,95221	0,92011
	Energy	0,44674	0,47085	0,40796	0,35118	0,40686	0,49658	0,22408	0,38124
	Homogeneity	0,92732	0,97195	0,93284	0,96033	0,94845	0,96825	0,90950	0,94381

Таблица 4 Векторы параметров на тестовых реальных изображениях пшеницы

ТЕСТОВЫЕ ПАРАМЕТРЫ

БОЛЕЗНЬ		1	2	3	4	5	6	7	8	Неизвестно
R	Contrast	0,01216	0,00338	0,00761	0,00267	0,00508	0,00569	0,00603	0,00470	0,00277
	Correlation	0,82058	0,96374	0,95313	0,97296	0,94501	0,94818	0,97359	0,94441	0,91563
	Energy	0,14972	0,27834	0,14796	0,22503	0,26376	0,15432	0,41224	0,23749	0,45479
	Homogeneity	0,78687	0,92207	0,85175	0,94177	0,88534	0,87031	0,90630	0,89703	0,93222
G	Contrast	0,01222	0,00353	0,00624	0,00274	0,00525	0,00553	0,00595	0,00442	0,00345
	Correlation	0,82961	0,96802	0,95193	0,95632	0,92457	0,95805	0,96752	0,93757	0,89513
	Energy	0,13552	0,25257	0,19652	0,30110	0,21352	0,15830	0,43052	0,22669	0,32478
	Homogeneity	0,78028	0,91904	0,87009	0,94303	0,88218	0,87549	0,90767	0,89919	0,91550
B	Contrast	0,01091	0,00291	0,00690	0,00220	0,00527	0,00539	0,00783	0,00462	0,00223
	Correlation	0,79723	0,95110	0,93528	0,94005	0,90041	0,93892	0,97635	0,94104	0,99110
	Energy	0,17365	0,32307	0,18482	0,38455	0,21951	0,18234	0,12167	0,20570	0,54630
	Homogeneity	0,79164	0,93110	0,86025	0,95088	0,87699	0,87713	0,85546	0,89287	0,94530
RG	Contrast	0,00341	0,00024	0,00000	0,00022	0,00116	0,00000	0,00003	0,00097	0,00000
	Correlation	0,85098	1,00000	1,00000	1,00000	0,94188	1,00000	1,00000	0,84866	1,00000
	Energy	0,44966	0,91085	1,00000	0,87699	0,47762	0,99946	0,99331	0,71334	1,00000
	Homogeneity	0,91871	0,99415	1,00000	0,99467	0,97168	0,99993	0,99928	0,97617	1,00000
RB	Contrast	0,00630	0,00134	0,00250	0,00174	0,00265	0,00105	0,00404	0,00283	0,00232
	Correlation	0,89331	0,94975	0,90563	0,96932	0,93860	0,93328	0,95427	0,94358	0,96883
	Energy	0,19384	0,45947	0,42493	0,30380	0,35535	0,55678	0,20865	0,31808	0,34648
	Homogeneity	0,85626	0,96725	0,93947	0,95774	0,93529	0,97428	0,91125	0,93202	0,94310
GB	Contrast	0,00382	0,00126	0,00279	0,00142	0,00243	0,00085	0,00420	0,00185	0,00255
	Correlation	0,78082	0,95400	0,89641	0,93648	0,87054	0,95219	0,95125	0,92252	0,97182
	Energy	0,42975	0,44933	0,42197	0,49985	0,47726	0,52660	0,21201	0,46175	0,35524
	Homogeneity	0,90639	0,96903	0,93281	0,96609	0,94070	0,97937	0,90731	0,95504	0,93755

\\(Grade)\\Testing\\ProgramDiagnostic\\Debug\\ProgramDiagnostic.exe

Data loaded:

R	Contrast:	0.00739083	0.00416669	0.00660944	0.00309369	0.00555375	0.00536118	0.00598864	0.00507677	0.00337584
R	Correlation:	0.076288	0.06073	0.051758	0.077657	0.037707	0.060835	0.074063	0.04825	0.06374
R	Energy:	0.19492	0.279356	0.119421	0.216139	0.250338	0.176768	0.399722	0.185873	0.278344
R	Homogeneity:	0.041222	0.007433	0.052082	0.040437	0.094552	0.083141	0.091957	0.082279	0.092207
G	Contrast:	0.00746357	0.00431503	0.00568162	0.00317792	0.00574819	0.00532859	0.00642086	0.00468484	0.00352966
G	Correlation:	0.09802	0.074321	0.042638	0.068792	0.029161	0.066788	0.071873	0.036734	0.060815
G	Energy:	0.178644	0.242773	0.1556	0.277785	0.221406	0.171944	0.330993	0.198767	0.252569
G	Homogeneity:	0.038006	0.006235	0.06968	0.04002	0.090187	0.084601	0.012352	0.088083	0.0919038
B	Contrast:	0.0070139	0.00355773	0.00600466	0.00208737	0.00544507	0.0051673	0.0027805	0.00442467	0.00291303
B	Correlation:	0.060021	0.057403	0.07987	0.052862	0.062845	0.056078	0.067125	0.029427	0.051104
B	Energy:	0.210083	0.310018	0.105224	0.31838	0.254092	0.194247	0.109195	0.222088	0.323866
B	Homogeneity:	0.042906	0.010652	0.062951	0.043708	0.091436	0.080443	0.060779	0.094692	0.0931105
RG	Contrast:	0.00250712	0.000226081	0	0.000338504	0.00110203	4.11e-07	0.000352087	0.00143007	0.000230734
RG	Correlation:	0.010106	1	1	1	0.030163	1	1	0.071049	1
RG	Energy:	0.0400372	0.040277	1	0.000031	0.0542408	0.009996	0.069853	0.071356	0.010040
RG	Homogeneity:	0.030663	0.090446	1	0.091705	0.071620	0.009999	0.091301	0.064736	0.004151
RB	Contrast:	0.00452069	0.00114546	0.00220106	0.00106074	0.00220041	0.00120041	0.00475559	0.0037173	0.00133763
RB	Correlation:	0.025046	0.023714	0.007402	0.075093	0.046622	0.026851	0.058126	0.039304	0.049747
RB	Energy:	0.213002	0.071109	0.453783	0.271306	0.362043	0.541744	0.171166	0.24136	0.459469
RB	Homogeneity:	0.001333	0.071963	0.04419	0.056399	0.046617	0.068164	0.002205	0.091136	0.067246
GB	Contrast:	0.0020061	0.00114731	0.00275051	0.00166683	0.00212731	0.00129605	0.00440808	0.00209661	0.00126435
GB	Correlation:	0.04671	0.063084	0.091084	0.062573	0.09171	0.042536	0.052206	0.020111	0.054003
GB	Energy:	0.446739	0.740853	0.407955	0.351179	0.406864	0.406581	0.224083	0.381242	0.44933
GB	Homogeneity:	0.027324	0.071954	0.032836	0.060334	0.04845	0.060247	0.009405	0.043009	0.060033

Рисунок 4. 1 Загруженные данные системой для диагностики

В рисунке 4.1, каждая строка соответствует значениям математических ожиданий параметров Харалика для каждой цветовой компоненты R, G, B, RG, RB, GB, и каждый столбец соответствует каждой болезни. Всего 8 болезней, и последний столбец - это вектор параметров Харалика исходного тестового изображения.

sA:	0.513068	0.595236	0.558783	0.576086	0.548891	0.58094	0.560587	0.535506
sB:	0.590647							
skoA*2:	3.60155	4.10371	4.21151	4.2702	3.836	4.27975	4.27114	3.91418
skoB*2:	4.1229							
Membership Function:								
R Contrast:	0.095985	0.099209	0.096766	0.099718	0.097822	0.098015	0.097387	0.098209
R Correlation:	0.012538	0.096089	0.088018	0.086083	0.073967	0.096294	0.080678	0.076784
R Energy:	0.016576	0.090808	0.041078	0.037795	0.071904	0.008424	0.078622	0.007520
R Homogeneity:	0.019152	0.085363	0.030012	0.081633	0.072482	0.061071	0.097087	0.060200
G Contrast:	0.096066	0.099215	0.097848	0.099648	0.097781	0.098201	0.097103	0.099845
G Correlation:	0.022805	0.093694	0.074623	0.099223	0.061146	0.098773	0.096142	0.098719
G Energy:	0.026075	0.090204	0.003031	0.074784	0.068836	0.019375	0.021577	0.046197
G Homogeneity:	0.018968	0.087197	0.050641	0.079018	0.071149	0.065562	0.093314	0.069949
B Contrast:	0.095899	0.099355	0.096098	0.099774	0.097468	0.097746	0.094635	0.098488
B Correlation:	0.008917	0.093702	0.078766	0.098242	0.05174	0.095026	0.08398	0.078323
B Energy:	0.087617	0.092952	0.042158	0.095315	0.031026	0.071181	0.078613	0.089022
B Homogeneity:	0.011882	0.087548	0.031847	0.087317	0.060332	0.055339	0.029674	0.063588
RG Contrast:	0.097672	0.099988	0.099761	0.0999	0.099077	0.099762	0.099886	0.098799
RG Correlation:	0.010196	1	1	1	0.036163	1	1	0.071849
RG Energy:	0.05321	0.070576	0.010848	0.087983	0.0321	0.010888	0.095005	0.066508
RG Homogeneity:	0.044512	0.099705	0.094151	0.097554	0.077477	0.094161	0.09723	0.070585
RB Contrast:	0.096089	0.099808	0.099057	0.099468	0.099038	0.099962	0.096582	0.09762
RB Correlation:	0.075299	0.073967	0.057735	0.074712	0.096875	0.077104	0.09162	0.089557
RB Energy:	0.075353	0.08836	0.094314	0.081837	0.003474	0.017725	0.071697	0.078191
RB Homogeneity:	0.024087	0.095283	0.076944	0.089153	0.07937	0.099002	0.093059	0.044114
GB Contrast:	0.098278	0.099883	0.098514	0.099598	0.099137	0.099968	0.096859	0.098968
GB Correlation:	0.082707	0.090119	0.037981	0.091429	0.063098	0.088533	0.098204	0.066109
GB Energy:	0.09741	0.078477	0.058626	0.081849	0.057534	0.052749	0.077453	0.093192
GB Homogeneity:	0.058291	0.097079	0.063083	0.091301	0.079418	0.099214	0.040462	0.074777

Рисунок 4. 2 Результат вычисления функций принадлежности

```

*****
Voting Parameter:
R Contrast: 1 1 1 1 1 1 1 1
R Correlation: 0 1 1 1 1 1 1 1
R Energy: 0 1 0 0 1 0 0 0
R Homogeneity: 0 1 0 1 1 1 1 1
G Contrast: 1 1 1 1 1 1 1 1
G Correlation: 0 1 1 1 1 1 1 1
G Energy: 0 1 0 1 1 0 0 0
G Homogeneity: 0 1 1 1 1 1 1 1
B Contrast: 1 1 1 1 1 1 1 1
B Correlation: 0 1 1 1 1 1 1 1
B Energy: 0 1 0 1 0 0 0 0
B Homogeneity: 0 1 0 1 1 1 0 1
RG Contrast: 1 1 1 1 1 1 1 1
RG Correlation: 0 1 1 1 0 1 1 0
RG Energy: 0 1 0 0 0 0 1 0
RG Homogeneity: 0 1 1 1 1 1 1 1
RB Contrast: 1 1 1 1 1 1 1 1
RB Correlation: 1 1 1 1 1 1 1 1
RB Energy: 0 0 1 0 0 0 0 0
RB Homogeneity: 0 1 1 1 1 1 0 0
GB Contrast: 1 1 1 1 1 1 1 1
GB Correlation: 0 1 0 1 1 1 1 1
GB Energy: 1 1 1 0 1 1 0 0
GB Homogeneity: 1 1 1 1 1 1 0 1
*****
Voting Component Before:
R 1 4 2 3 4 3 3 3
G 1 4 3 4 4 3 3 3
B 1 4 2 4 3 3 2 3
RG 1 4 3 3 2 3 4 2
RB 2 3 4 3 3 3 2 2
GB 3 4 3 3 4 4 2 3
*****
Voting Component After:
R 0 1 0 0 1 0 0 0
G 0 1 0 1 1 0 0 0
B 0 1 0 1 0 0 0 0
RG 0 1 0 0 0 0 1 0
RB 0 0 1 0 0 0 0 0
GB 0 1 0 0 1 1 0 0
*****

```

Рисунок 4. 3 Результат бинаризации и голосования

Использовать загруженные значения, мы вычисляем значения функции принадлежности параметров тестового изображения. Порог для бинаризации функции принадлежности равен 0.95. Все значения функции принадлежности больше чем

0.95, соответствуют 1, а меньше соответствует 0. По этому правилу, мы получаем результат голосования по каждому параметру. Комбинируем эти голосования по цветовым компонентам и максимальное голосования соответствует единице, остальные нулю, создается результат голосования по каждому цветовому компоненту.

Комбинируем результаты голосования по цветовому компоненту, получается окончательное голосование. И здесь, предсказательная болезнь имеет максимальное голосование.

```
Voting Global:
0 5 1 2 3 1 1 0
*****
Input: 2 | Predict: 2
```

Рисунок 4. 4 Итоговый голос и результат диагностики

Если бы выбирается только болезнь, имеющая максимальное голосование, то система не может распознавать неизвестное изображение, и не достигается надежную достоверность. Поэтому результат голосования является доверенным, который можем уверить, только если максимальное голосование больше чем 4. То есть больше половины всех голосований, голосует одну болезнь.

```
Voting Global:
0 2 2 3 3 2 1 2
*****
Input: 9 | Predict: unknown
```

Рисунок 4. 5 Результат диагностики неизвестной болезни

Например, в рисунке 4.5, на входе попадает изображение 9-ой болезни, но система может распознавать только 8 болезней. То в результат голосования, максимальное голосование равно 3, и является не доверенным результатом. Поэтому система выводит неизвестный результат.

В остальных случаях, все изображения в тестовом наборе были успешно распознавание по этому процессу.

ЗАКЛЮЧЕНИЕ

В ходе выполнения этой работы были получены следующие результаты:

1. Рассмотрены и проанализированы известные способы диагностики болезней растений по изображениям листьев.
2. Изучены методы обработки изображений, предложен и реализован алгоритм нормализации изображений методом фильтрации Отсу и сегментации изображения алгоритмом по водоразделам Watershed, с применением морфологических операций, поворота и масштабирования.
3. Изучена теория нечеткой логики и предложен метод диагностики болезней растений, основанный на нечеткой логике, по нормализованным изображениям листьев, основанный на вычислении 4-х текстурных характеристик Харалика изображений,, вычисленных с использованием матриц GLCM для 6-ти цветовых компонентов изображений: R, G, B, RG, RB, GB.
4. Рассмотрены принципы работы нейронных сетей. Для реализации диагностики болезней растений по нормализованным изображениям листьев, основанной на текстурных характеристиках, выбрана сеть прямого распространения.
5. Доказано путем моделирования,, что можно достигнуть достоверности распознавания около 95% за счет бинаризации результатов и мажоритарного голосования.
6. Так же путем моделирования показана зависимость точности от объема данных метода нечеткой логики и метода нейронной сети. Отсюда, определены нужный объем данных для каждого метода.

7. Произведена экспериментальная проверка метода диагностики на реальных изображениях листьев пшеницы, подтвердившая результаты, полученные при моделировании.
8. Разработано программное обеспечение для компьютерной диагностики заболеваний растений по текстурным характеристикам изображений листьев.

Разработанная система диагностики растений по изображениям листьев нами, имеет два способа распознавания типа заболевания по изображениям. Метод нечеткой логики и метод нейронной сети, оба метода классификации основаны на текстурных характеристиках – именно характеристики Харалика, вычисленные матрицей смежности GLCM для шести цветовых компонентов: R,G,B,RG,GB,RB, из нормализованного изображения, классическим методом обработки изображения с помощью открытой библиотеки OpenCV чтобы удалять неинформативную часть в изображении.

По сравнению с известными, так как система имеется два способа диагностики, и один из них, метод нечеткой логики, можем работать с небольшим набором данных, и достигается хорошую точность. Результат доказывается в разделе моделирования. Не как известные системы, наша система уже может достигать лучший результат с начала применения, когда набор данных еще небольшой. Но метод нечеткой логики так же имеет недостаток, это когда база данных увеличивается, количество вычислений так же пропорционально увеличивается, потому что чтобы улучшить результат, каждый раз система будет снова вычислить эталонное описание, тогда требует много вычислений. Эту проблему решает второй метод - метод нейронной сети. Метод нейронной сети применяется с момента, когда система собирает достаточный объем данных, количество изображений по каждой болезни примерно 150. Когда объем

набора данных большой, мы переучиваем систему методом нейронной сети. Тогда мы можем использовать эту обученную модель нейронной сети чтобы классификацию типы заболевания. Это модель не как модель нечеткой логики, количество вычислений на этапе применения которого зависит от только структуры модели, и не изменяется, когда набор данных увеличивается.

Разработанная система нами были проверены на набор данных пшеницы, но так же предлагаем применить для других типов растений, заболевания которых отображаются на своих листьях, например кукуруза.... Система может применяться по следующим принципам: в начале, когда система еще не имеет большой объем данных, мы можем одновременно применять нечеткую логику для диагностики, и собираем данные, потом, когда данные собираются достаточно, мы можем применять нейронную сеть, чтобы уменьшить количество вычислений внутри системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Денисюк В.С. Алгоритмы выделения особенностей на изображениях с целью классификации заболеваний растений // Конструирование и оптимизация параллельных программ - Новосибирск, 2008. №16. с.171-182.
2. Койшибаев М. Болезни пшеницы // Продовольственная и сельскохозяйственная организация (ФАО), Анкара, 2018. с 365.
3. Лабинский А.Ю. Использование нечеткой логики в решении задач классификации. // Научно-аналитический журнал «Вестник Санкт-Петербургского университета Государственной противопожарной службы МЧС России», 2018. – 8с.
4. Сергиенко А.Б. Цифровая обработка сигналов. -СПб.: Изд-во Питер, 2013г. – 608 с.
5. Григорий Сапунов. Введение в архитектуры нейронных сетей // Конференции Олега Бунина (Онтико), 10/2017. – URL: <https://habr.com/ru/company/oleg-bunin/blog/340184/>.
6. Николай Паклин. Нечеткая логика – математические основы, 2005. URL: <https://basegroup.ru/community/articles/fuzzylogic-math>.
7. Hall D.; McCool C.; Dayoub F.; Sunderhauf N.; Upcroft B. Evaluation of features for leaf classification in challenging conditions // In Proceedings of the 2015 IEEE Winter Conference on Applications of Computer Vision, Waikoloa Beach, HI, USA, 6–8 January 2015; 797–804.
8. Kussul N., Lavreniuk M., Skakun S., Shelestov A., Deep learning classification of land cover and crop types using remote sensing data // IEEE Geosci. Remote Sens. Lett. 2017, 14, 778–782.
9. Mortensen A.K.; Dyrmann M.; Karstoft H.; Jørgensen R.N.; Gislum R. Semantic segmentation of mixed crops using deep convolutional neural network// In Proceedings of the CIGR-

- AgEng Conference, Aarhus, Denmark, 26–29 June 2016; Abstracts and Full Papers. pp. 1–6.
10. Muhammad Hammad Saleem, Johan Potgieter and Khalid Mahmood Arif. Plant Disease Detection and Classification by Deep Learning // *Plants* 2019, vol. 1, pp. 2-3.
 11. Liu T., Wu W., Chen W., Sun C., Zhu X., Guo W. Automated image-processing for counting seedlings in a wheat field // *Precis. Agric.* 2016, 17, 392–406.
 12. Professor Patil Ashish, Patil Tanuja. Survey on Detection and Classification of Plant Leaf Disease in Agriculture Enviroment // *NCIAR CSE – 2017. Vol 4.* 137-139.
 13. Rußwurm M.; Körner M. Multi-temporal land cover classification with long short-term memory neural networks // *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* 2017, 42, 551.
 14. Rebetez J.; Satizábal H. F.; Mota M.; Noll D.; Büchi L.; Wendling M.; Cannelle B.; Pérez-Urbe A.; Burgos S. Augmenting a convolutional neural network with local histograms - A case study in crop classification from high-resolution UAV imagery // *In Proceedings of the ESANN, Bruges, Belgium, 27–29 April 2016.*
 15. R. M. Haralick. Statistical and structural approaches to texture // *Proceedings of the IEEE*, 1979. Vol. 67, no. 5, pp. 768–804.
 16. Sun Y.; Liu Y.; Wang G.; Zhang H. Deep learning for plant identification in natural environment // *Comput. Intell. Neurosci.* 2017, 736-1042.
 17. Kuan-Man Xu. Using the Bootstrap Method for a Statistical Significance Test of Differences between Summary Histograms // *NASA Langley Research Center, Hampton, VA.* [URL: ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080015431.pdf](https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080015431.pdf)
 18. Mryka Hall-Beyer, Ph.D. GLCM Texture: A tutorial. https://prism.ucalgary.ca/bitstream/handle/1880/51900/texture%20tutorial%20v%203_0%20180206.pdf?sequence=11&isAllowed=y

ПРИЛОЖЕНИЕ 1: КЛАСС НОРМАЛИЗАЦИИ ИЗОБРАЖЕНИЯ

```
#pragma once
#ifndef PREPROCESS_IMAGE_H
#define PREPROCESS_IMAGE_H

#include <opencv2/opencv.hpp>
#include <math.h>
#include <vector>

#define DEBUG
using namespace cv;

class Preprocess
{
public:
    Preprocess();
    ~Preprocess();

    Mat* processing(); //
    preprocessing image
    void updateImg(Mat& new_img); // update source image
    Mat* getSourceImg() { return src; };

private:
    Mat* src; // source image
    Mat* dst; // destination image

    Mat watershedSegmentation(Mat &img, int centreW, int
centreH);
    Mat* rotate(Mat &img, double angle, Point &center);
    Mat* crop(Mat& img); // size 300x100

    const Scalar red = Scalar(0, 0, 255);
    const Scalar green = Scalar(0, 255, 0);
    const Scalar blue = Scalar(255, 0, 0);
};

class WatershedSegmenter {
private:
    cv::Mat markers;
public:
    bool setMarkers(cv::Mat& markerImage)
    {
        if (countNonZero(markerImage) < 1)
            return false;
        markerImage.convertTo(markers, CV_32S);
        return true;
    }
}
```

```

        cv::Mat process(cv::Mat &image)
        {
            if (image.rows == 0 || image.cols == 0)
                return image;
            cv::watershed(image, markers);
            markers.convertTo(markers, CV_8U);
            return markers;
        }
};

double getAvgPixel(Mat img);
#endif // !PREPROCESS_IMAGE_H

#include "Preprocess.h"

double getAvgPixel(Mat img)
{
    double res = 0, num = 0;
    for (int i = 0; i < img.rows; i++)
        for (int j = 0; j < img.cols; j++)
            if (img.at<uchar>(i, j) != 0) {
                res += img.at<uchar>(i, j);
                num = num + 1.0;
            }
    return res / num;
}

Preprocess::Preprocess()
{
    // create black image
    src = new Mat(600, 800, CV_8UC1, Scalar(0, 0, 0));
    dst = new Mat();
}

Preprocess::~Preprocess()
{
    if (src != nullptr)
        delete src;
    if (dst != nullptr)
        delete dst;
}

Mat * Preprocess::processing()
{
    // preprocess
    Mat img = src->clone(), drawing = src->clone();
    double angle = 0;
    Point center(0, 0);
    std::vector<std::vector<Point>> contours;
    std::vector<Vec4i> hierarchy;

```

```

pyrMeanShiftFiltering(img, img, 3, 8); // 8,3
#ifdef DEBUG
    imshow("MeanShiftFiltering", img);
#endif // DEBUG

// Create a kernel that we will use to sharpen our image
Mat kernel = (Mat_<float>(3, 3) <<
    0, -1, 0,
    -1, 5, -1,
    0, -1, 0);
Mat imgLaplacian;
filter2D(img, imgLaplacian, CV_32F, kernel);
Mat sharp;
img.convertTo(sharp, CV_32F);
img = sharp - imgLaplacian;
// convert back to 8bits gray scale
img.convertTo(img, CV_8UC3);
imgLaplacian.convertTo(imgLaplacian, CV_8UC3);
#ifdef DEBUG
    imshow("Laplace Filtered Image", imgLaplacian);
    imshow("Sharped Image", img);
#endif // DEBUG

Mat dest = watershedSegmentation(drawing, img.cols / 4,
img.rows / 4);
#ifdef DEBUG
    imshow("SegmentationColor", dest);
#endif // DEBUG

cvtColor(dest, dest, CV_RGB2HSV);
#ifdef DEBUG
    imshow("HSV", dest);
#endif // DEBUG
Mat hsv[3];
split(dest, hsv);
double vh = getAvgPixel(hsv[0]), vs = getAvgPixel(hsv[1]), vv
= getAvgPixel(hsv[2]);
inRange(dest, Scalar(vh, vs, vv), Scalar(255, 255, 255),
dest);
Mat structuringElement = getStructuringElement(MORPH_ELLIPSE,
Size(3, 3));
morphologyEx(dest, dest, MORPH_CLOSE, structuringElement);
#ifdef DEBUG
    imshow("Range", dest);
#endif // DEBUG

findContours(dest, contours, hierarchy, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
double maxS = 0;
int idM = 0;

```



```

        for (int i = 0; i < contours.size(); i++) {
            if (contourArea(contours[i]) > maxS) {
                idM = i;
                maxS = contourArea(contours[i]);
            }
        }
        RotatedRect minBound(Point2f(0, 0), Size2f(0, 0), 0);
        if (contours.size() > 0) {
            drawContours(drawing, contours, idM, red, 2);
            minBound = fitEllipse(contours[idM]);
        }
        if (maxS < (double)(dest.rows*dest.cols) / 100.0) {
            dest = watershedSegmentation(img, img.cols / 8,
img.rows / 8);
            cvtColor(dest, img, COLOR_BGR2GRAY);
#ifdef DEBUG
                imshow("Gray", img);
#endif // DEBUG

            Mat thresh;
            double vthresh = getAvgPixel(img);
            threshold(img, thresh, vthresh, 255,
CV_THRESH_BINARY | CV_THRESH_OTSU);
#ifdef DEBUG
                imshow("Theshold", thresh);
#endif // DEBUG

            Mat structuringElement =
getStructuringElement(MORPH_ELLIPSE, Size(img.cols / 8, img.rows /
8));
            morphologyEx(thresh, thresh, MORPH_CLOSE,
structuringElement);
#ifdef DEBUG
                imshow("Grouping", thresh);
#endif // DEBUG

            if (contours.size() > 0){
                contours.clear();
                hierarchy.clear();
            }
            findContours(thresh, contours, hierarchy,
CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
            maxS = 0;
            idM = 0;
            for (int i = 0; i < contours.size(); i++) {
                if (contourArea(contours[i]) > maxS) {
                    idM = i;
                    maxS = contourArea(contours[i]);
                }
            }
        }
    }
}

```

```

        drawContours(drawing, contours, idM, green, 2);
        minBound = fitEllipse(contours[idM]);
    }
    ellipse(drawing, minBound, green, 2);
    Point2f rect_points[4];
    minBound.points(rect_points);
    for (int j = 0; j < 4; j++)
        line(drawing, rect_points[j], rect_points[(j + 1) %
4], blue, 2);
#ifdef DEBUG
    imshow("Drawing", drawing);
#endif // DEBUG

    center = minBound.center;
    if (minBound.angle > 90)
        angle = minBound.angle - 180;
    else
        angle = minBound.angle;

    dst = rotate(*src, angle, center);
#ifdef DEBUG
    imshow("Rotate", *dst);
#endif // DEBUG
    Range drows(max((int)(center.y - minBound.size.height / 2),
0), min((int)(center.y + minBound.size.height / 2), img.rows));
    Range dcols(max((int)(center.x - minBound.size.width / 2),
0), min((int)(center.x + minBound.size.width / 2), img.cols));
    *dst = (*dst)(drows, dcols);
    resize(*dst, *dst, Size(100, 300));
    return dst;
}

```

```

void Preprocess::updateImg(Mat & new_img)
{
    *src = new_img;
}

```

```

Mat Preprocess::watershedSegmentation(Mat & img, int centreW, int
centreH)
{
    // Watershed segmentation
    Mat markers(img.size(), CV_8U, cv::Scalar(-1));
    //top rectangle
    markers(Rect(0, 0, img.cols, 5)) = Scalar::all(1);
    //bottom rectangle
    markers(Rect(0, img.rows - 5, img.cols, 5)) = Scalar::all(1);
    //left rectangle
    markers(Rect(0, 0, 5, img.rows)) = Scalar::all(1);
    //right rectangle

```

```

        markers(Rect(img.cols - 5, 0, 5, img.rows)) = Scalar::all(1);
        //centre rectangle
        markers(Rect((img.cols / 2) - (centreW / 2), (img.rows / 2) -
        (centreH / 2), centreW, centreH)) = Scalar::all(2);
        markers.convertTo(markers, CV_BGR2GRAY);
        #ifdef DEBUG
            imshow("markers", markers);
        #endif // DEBUG
        WatershedSegmenter segmenter;
        segmenter.setMarkers(markers);
        // For grayscale sharpened image
        Mat dest;
        cv::Mat wshedMask = segmenter.process(img);
        cv::Mat mask;
        convertScaleAbs(wshedMask, mask, 1, 0);
        threshold(mask, mask, 1, 255, THRESH_BINARY);
        bitwise_and(img, img, dest, mask);
        dest.convertTo(dest, CV_8U);
        #ifdef DEBUG
            imshow("Watershed Segmentation", dest);
        #endif // DEBUG
        return dest;
    }

    Mat * Preprocess::rotate(Mat & img, double angle, Point & center)
    {
        Mat* res = new Mat();
        Mat* rot = new Mat();
        *rot = getRotationMatrix2D(center, angle, 1.0);
        warpAffine(img, *res, *rot, img.size());
        delete rot;
        return res;
    }

    Mat * Preprocess::crop(Mat & img)
    {
        return &img;
    }

```

ПРИЛОЖЕНИЕ 2: КЛАСС ВЫЧИСЛЕНИЯ ПАРАМЕТРОВ ХАРАЛИКА

```
#pragma once
#include <opencv2/opencv.hpp> // for using library opencv
#include <fstream> // for working with file (write output)

using namespace cv;
// Gray level - 4, 8, 16
enum GrayLevel
{
    GRAY_4,
    GRAY_8,
    GRAY_16
};
enum ChannelRGB
{
    CHANNEL_R,
    CHANNEL_G,
    CHANNEL_B,
    CHANNEL_RG,
    CHANNEL_RB,
    CHANNEL_GB
};

class GLCM
{
public:
    GLCM();
    GLCM(GrayLevel gl);
    ~GLCM();

    Mat getImgByChannel(Mat &img, ChannelRGB cn);

    void calculateMatrix(Mat &img, int delta_row, int delta_col);
    void normalizeMatrix();
    void calculateFeatures();

    double** getMatrix();
    void printMatrix();
    double* getFeatures_Haralick();
    void printFeatures_Haralick(std::ofstream &outFile);
    int getGLevel() { return gLevel; };

    void reset();

private:
    double** matrix;
    double* features; // [6] = contrast, correlation, energy,
    homogeneity, entropy, Inverse Difference Moment
```

```

        int gLevel; // size of matrix - gray level
        double sumValue;
        char nameFeatures[6][50] = { "      -Contrast: ", "      -
Correlation: ", "      -Energy: ", "      -Homogeneity: ",
        "      -Entropy: ", "      -Inverse Difference Moment:
" };
    };
};

#include "GLCM.h"
GLCM::GLCM()
{
    gLevel = 8;
    matrix = new double *[gLevel];
    for (int i = 0; i < gLevel; i++) {
        matrix[i] = new double[gLevel];
    }
    features = new double[6];
}

GLCM::GLCM(GrayLevel gl)
{
    switch (gl)
    {
        case GRAY_4:
            gLevel = 4;
            break;
        case GRAY_8:
            gLevel = 8;
            break;
        case GRAY_16:
            gLevel = 16;
            break;
        default:
            gLevel = 8;
            break;
    }
    matrix = new double *[gLevel];
    for (int i = 0; i < gLevel; i++) {
        matrix[i] = new double[gLevel];
    }
    features = new double[6];
}

GLCM::~~GLCM()
{
    if (matrix) {
        for (int i = 0; i < gLevel; i++)
            delete matrix[i];
        delete matrix;
    }
    if (features)

```

```

        delete features;
    }

Mat GLCM::getImgByChannel(Mat & img, ChannelRGB cn)
{
    Mat bgr[3];
    split(img, bgr);
    switch (cn)
    {
        case CHANNEL_R:
            return bgr[2];
            break;
        case CHANNEL_G:
            return bgr[1];
            break;
        case CHANNEL_B:
            return bgr[0];
            break;
        case CHANNEL_RG:
            return (bgr[2] - bgr[1]);
            break;
        case CHANNEL_RB:
            return (bgr[2] - bgr[0]);
            break;
        case CHANNEL_GB:
            return (bgr[1] - bgr[0]);
            break;
        default:
            std::cout << "Incorrect type of channel! \n";
            exit(1);
    }
}

void GLCM::calculateMatrix(Mat & img, int delta_row, int delta_col)
{
    sumValue = 0;
    int coefficient = 32;
    switch (gLevel)
    {
        case 4:
            coefficient = 64;
            break;
        case 8:
            coefficient = 32;
            break;
        case 16:
            coefficient = 16;
            break;
        default:
            break;
    }
}

```

```

    }
    int imatrix, jmatrix;
    int newi, newj;
    for (int i = 0; i < img.rows; i++) {
        for (int j = 0; j < img.cols; j++) {
            newi = i + delta_row;
            newj = j + delta_col;
            if (newi < img.rows && newi >= 0 && newj <
img.cols && newj >= 0) {
                imatrix = (int)img.at<uchar>(i, j) /
coefficient;
                jmatrix = (int)img.at<uchar>(newi,
newj) / coefficient;
                matrix[imatrix][jmatrix] += 1.0;
                sumValue += 1.0;
            }
        }
    }
}
void GLCM::normalizeMatrix()
{
    for (int i = 0; i < gLevel; i++) {
        for (int j = 0; j < gLevel; j++) {
            matrix[i][j] /= sumValue;
        }
    }
}

void GLCM::calculateFeatures()
{
    // contrast
    features[0] = 0;
    for (int i = 0; i < gLevel; i++) {
        for (int j = 0; j < gLevel; j++) {
            features[0] += pow((i - j), 2)*matrix[i][j];
        }
    }
    //features[0] /= pow(gLevel - 1, 2);
    // correlation
    features[1] = 0;
    double* mu_u = new double[gLevel + 1];
    double* si_u = new double[gLevel + 1];
    double* mu_v = new double[gLevel + 1];
    double* si_v = new double[gLevel + 1];

    mu_u[gLevel] = 0;
    for (int i = 0; i < gLevel; i++) {
        mu_u[i] = 0.0;
        for (int j = 0; j < gLevel; j++) {
            mu_u[i] += i * matrix[i][j];

```

```

        }
        mu_u[gLevel] += mu_u[i];
    }
    mu_v[gLevel] = 0;
    for (int j = 0; j < gLevel; j++) {
        mu_v[j] = 0;
        for (int i = 0; i < gLevel; i++) {
            mu_v[j] += j * matrix[i][j];
        }
        mu_v[gLevel] += mu_v[j];
    }
    si_u[gLevel] = 0;
    for (int i = 0; i < gLevel; i++) {
        si_u[i] = 0;
        for (int j = 0; j < gLevel; j++) {
            si_u[i] += matrix[i][j] * pow((i -
mu_u[gLevel]), 2);
        }
        //si_u[i] = sqrt(si_u[i]);
        si_u[gLevel] += si_u[i];
    }
    si_v[gLevel] = 0;
    for (int j = 0; j < gLevel; j++) {
        si_v[j] = 0;
        for (int i = 0; i < gLevel; i++) {
            si_v[j] += matrix[i][j] * pow((j -
mu_v[gLevel]), 2);
        }
        //si_v[j] = sqrt(si_v[j]);
        si_v[gLevel] += si_v[j];
    }

    for (int i = 0; i < gLevel; i++) {
        for (int j = 0; j < gLevel; j++) {
            if (matrix[i][j] != 0)
                features[1] += matrix[i][j] * (i -
mu_u[gLevel]) * (j - mu_v[gLevel]) / sqrt(si_u[gLevel] * si_v[gLevel]);
        }
    }
    //features[1] = (features[1] + 1) / 2;
    // energy
    features[2] = 0;
    for (int i = 0; i < gLevel; i++) {
        for (int j = 0; j < gLevel; j++) {
            features[2] += pow(matrix[i][j], 2);
        }
    }
    // homogeneity
    features[3] = 0;
    for (int i = 0; i < gLevel; i++) {

```



```

        for (int j = 0; j < gLevel; j++) {
            features[3] += matrix[i][j] / (1 + abs(i -
j));
        }
    }
    // entropy
    features[4] = 0;
    for (int i = 0; i < gLevel; i++) {
        for (int j = 0; j < gLevel; j++) {
            if (matrix[i][j] != 0)
                features[4] += matrix[i][j] *
log10(matrix[i][j]);
        }
    }
    features[4] = -features[4];
    // Inverse Difference Moment
    features[5] = 0;
    for (int i = 0; i < gLevel; i++) {
        for (int j = 0; j < gLevel; j++) {
            features[5] += matrix[i][j] / (1 + pow(i -
j, 2));
        }
    }

    delete mu_u, mu_v, si_u, si_v;
}

double ** GLCM::getMatrix()
{
    return matrix;
}

void GLCM::printMatrix()
{
    for (int i = 0; i < gLevel; i++) {
        for (int j = 0; j < gLevel; j++) {
            std::cout << matrix[i][j] << "    ";
        }
        std::cout << std::endl;
    }
    std::cout << sumValue << std::endl;
}

double * GLCM::getFeatures_Haralick()
{
    return features;
}

void GLCM::printFeatures_Haralick(std::ofstream &outFile)
{
    std::cout << "  -Haralick's features: \n";
    for (int i = 0; i < 6; i++) {

```

```

        std::cout << nameFeatures[i] << features[i] <<
std::endl;
        outFile << features[i] << ", ";
    }
}
void GLCM::reset()
{
    for (int i = 0; i < gLevel; i++) {
        for (int j = 0; j < gLevel; j++) {
            matrix[i][j] = 0;
        }
    }
    for (int i = 0; i < 6; i++) {
        features[i] = 0;
    }
    sumValue = 0;
}

```

ПРИЛОЖЕНИЕ 3: КЛАСС ДИАГНОСТИКИ НА НЕЧЕТКОЙ ЛОГИКЕ

```
#pragma once
#include <iostream>
#include <vector>
#include <fstream>
#include <string>
#include <algorithm>
#include <iomanip>
using namespace std;
class Classifier {
public:
    Classifier();
    Classifier(int k_, int nd_);
    ~Classifier();

    void reset() {
        matrixA.clear();
        vectorB.clear();
        sA.clear();
        sB = 0;
        skoA.clear();
        skoB = 0;
        mf.clear();
        mfComponent.clear();
        votingComponentAfter.clear();
        votingComponentBefore.clear();
        votingParameter.clear();
        votingGlobal.clear();
    }

    void loadData(string path, vector<double> noise, double sko);
    void printData();
    void calculateS();
    void calculateSKO();
    void membershipFunction();
    int predict();

    vector<double> getVectorB() { return vectorB; };
    vector<double> getVectorS_A() { return sA; };
    vector<double> getVectorSKO_A() { return skoA; };
    double getS_B() { return sB; };
    double getSKO_B() { return skoB; };
private:
    vector<vector<double>> matrixA;
    vector<double> vectorB;

    vector<double> sA;
    double sB = 0;
```

```

double skoB = 0;
vector<double> skoA;

vector<double> mf;
vector<vector<double>> mfComponent;
vector<vector<int>> votingParameter;
vector<vector<int>> votingComponentBefore;
vector<vector<int>> votingComponentAfter;
vector<int> votingGlobal;

const double BT = 0.95; // коэффициент для порога бинаризации
const int NP = 24;      // количество параметров
Харалика
int k;                  // номер заболевания на входе
int nd;                 // количество болезней, имеющихсь в базе
данных
int method = 2;        // 1 or dif
bool testing = false;  // использовать шум ли
char nameFeatures[4][20] = { "Contrast: ", "Correlation: ",
"Energy: ", "Homogeneity: " };
char nameComponent[6][5] = { "R ", "G ", "B ", "RG ", "RB ",
"GB " };
};

#include "Classifier.h"
Classifier::Classifier()
{
    k = 1;
    nd = 8;
}
Classifier::Classifier(int k_, int nd_)
{
    k = k_;
    nd = nd_;
}
Classifier::~Classifier() {}
void Classifier::loadData(string path, vector<double> noise, double
sko)
{
    matrixA.clear();
    vectorB.clear();
    ifstream input;
    input.open(path);
    if (!input.is_open())
        cout << "Path is incorrect \n";
    string data = "";
    while (input.good())
    {
        vector<double> rws;
        for (int i = 0; i < nd; i++) {

```

```

        getline(input, data, ';');
        rws.push_back(atof(data.c_str()));
    }
    for (int i = 0; i <= nd; i++) {
        if (i != nd)
            getline(input, data, ';');
        else
            getline(input, data);
        if (i == (k - 1))

vectorB.push_back(atof(data.c_str()));
    }
    matrixA.push_back(rws);
}
input.close();
if (testing == true) {
    for (int i = 0; i < NP; i++)
        vectorB[i] = matrixA[i][k - 1] + sko *
noise[i];
}
}
void Classifier::printData()
{
    cout << "
Data loaded: \n";
    for (int i = 0; i < NP; i++) {
        int component = i / 4;
        int feature_id = i % 4;
        cout << setw(4) << nameComponent[component] <<
setw(15) << nameFeatures[feature_id];
        for (int j = 0; j < nd; j++)
            cout << setw(15) << matrixA[i][j] << " ";
        cout << " | " << vectorB[i] << endl;
    }
    cout << "
*****\n";
    cout << "sA: ";
    for (int i = 0; i < nd; i++)
        cout << setw(10) << sA[i] << " ";
    cout << "\nsB: " << setw(10) << sB << endl;
    cout << "
*****\n";
    cout << "skoA^2: ";
    for (int i = 0; i < nd; i++)
        cout << setw(10) << skoA[i] << " ";
    cout << "\nskoB^2: " << setw(10) << skoB << endl;
    cout << "
*****\n";
    cout << "Membership Function: \n";
    if (method == 1) {

```

```

        for (int i = 0; i < nd; i++)
            cout << mf[i] << " ";
        cout << endl;
    }
    else {
        for (int j = 0; j < NP; j++) {
            int component = j / 4;
            int feature_id = j % 4;
            cout << setw(4) << nameComponent[component]
<< setw(15) << nameFeatures[feature_id];
            for (int i = 0; i < nd; i++)
                cout << setw(10) <<
mfComponent[i][j] << " ";
            cout << endl;
        }
        cout << "
*****\n";
        cout << "Voting Parameter: \n";
        for (int j = 0; j < NP; j++) {
            int component = j / 4;
            int feature_id = j % 4;
            cout << setw(4) << nameComponent[component]
<< setw(15) << nameFeatures[feature_id];
            for (int i = 0; i < nd; i++)
                cout << votingParameter[i][j] << "
";
            cout << endl;
        }
        cout << "
*****\n";
        cout << "Voting Component Before: \n";
        for (int j = 0; j < 6; j++) {
            cout << setw(4) << nameComponent[j];
            for (int i = 0; i < nd; i++)
                cout << votingComponentBefore[i][j]
<< " ";
            cout << endl;
        }
        cout << "
*****\n";
        cout << "Voting Component After: \n";
        for (int i = 0; i < 6; i++) {
            cout << setw(4) << nameComponent[i];
            for (int j = 0; j < nd; j++)
                cout << votingComponentAfter[i][j]
<< " ";
            cout << endl;
        }
        cout << "
*****\n";

```

```

        cout << "Voting Global: \n";
        for (int i = 0; i < nd; i++)
            cout << votingGlobal[i] << " ";
        cout << endl;
    }
}

void Classifier::calculateS()
{
    sA.clear();
    for (int i = 0; i < nd; i++) {
        double s_a = 0;
        for (int j = 0; j < NP; j++) {
            s_a += matrixA[j][i];
        }
        sA.push_back(s_a / (double)NP);
    }
    sB = 0;
    for (int j = 0; j < NP; j++) {
        sB += vectorB[j];
    }
    sB = sB / (double)NP;
}

void Classifier::calculateSKO()
{
    skoA.clear();
    for (int i = 0; i < nd; i++) {
        double sko = 0;
        for (int j = 0; j < NP; j++) {
            sko += pow(matrixA[j][i] - sA[i], 2);
        }
        skoA.push_back(sko);
    }
    skoB = 0;
    for (int j = 0; j < NP; j++) {
        skoB += pow(vectorB[j] - sB, 2);
    }
}

void Classifier::membershipFunction()
{
    mf = vector<double>(nd, 0);
    for (int i = 0; i < nd; i++) {
        for (int j = 0; j < NP; j++) {
            mf[i] += ((matrixA[j][i] -
sA[i])*(vectorB[j] - sB));
        }
        mf[i] /= (sqrt(skoA[i] * skoB));
    }
    //////////// MF for component //////////
    for (int i = 0; i < nd; i++) {
        vector<double> p;

```

```

        for (int j = 0; j < NP; j++) {
            p.push_back(1.0 - abs(matrixA[j][i] -
vectorB[j]));
        }
        mfComponent.push_back(p);
    }
    for (int i = 0; i < nd; i++) {
        vector<int> p;
        for (int j = 0; j < NP; j++) {
            p.push_back(0);
            if (mfComponent[i][j] >= BT)
                p[j] = 1;
        }
        votingParameter.push_back(p);
    }
    for (int i = 0; i < nd; i++) {
        vector<int> p(6, 0);
        for (int j = 0; j < NP; j++) {
            p[j / 4] += votingParameter[i][j];
        }
        votingComponentBefore.push_back(p);
    }
    for (int i = 0; i < 6; i++) {
        vector<int> p(nd, 0);
        int maxV = 0;
        for (int j = 0; j < nd; j++)
            if (votingComponentBefore[j][i] > maxV)
                maxV = votingComponentBefore[j][i];
        for (int j = 0; j < nd; j++) {
            if (votingComponentBefore[j][i] == maxV &&
maxV > 1)
                p[j] = 1;
        }
        votingComponentAfter.push_back(p);
    }
    votingGlobal = vector<int>(nd, 0);
    for (int i = 0; i < nd; i++) {
        for (int j = 0; j < 6; j++)
            votingGlobal[i] +=
votingComponentAfter[j][i];
    }
}
int Classifier::predict()
{
    if (method == 1) {
        double id = 0, mfMax = 0;
        for (int i = 0; i < nd; i++) {
            if (mf[i] > mfMax) {
                id = i + 1;
                mfMax = mf[i];
            }
        }
    }
}

```



```

    }
    cout << "
*****\n";
    cout << "Input: " << k << " | Predict: ";
    if (mfMax >= 0.99) {
        cout << id << endl;
        return id;
    }
    else {
        cout << "unknown \n";
        return 0;
    }
}
else {
    int maxV = *max_element(votingGlobal.begin(),
votingGlobal.end()), id = 0;
    cout << "
*****\n";
    cout << "Input: " << k << " | Predict: ";
    if (maxV < 4) {
        cout << "unknown \n";
        return 0;
    }
    else {
        for (int i = 0; i < nd; i++)
            if (votingGlobal[i] == maxV)
                id = i + 1;
        cout << id << endl;
        return id;
    }
}
return 0;
}

```

ПРИЛОЖЕНИЕ 4: ПРОГРАММА ДИАГНОСТИКИ НА НЕЙРОННОЙ СЕТИ

```
import pandas as pd
import numpy as np
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dropout, Dense, Flatten
from keras.utils import to_categorical

pathToTrain =
'I:/GradEs/CompareNN_FL/MatlabProgram/DataTrainGenerated2.xlsx'
pathToTest =
'I:/GradEs/CompareNN_FL/MatlabProgram/TestGroupGenerated2.xlsx'

Ntrain = 1000 # Объем данных для обучения
print("Ntrain: ", Ntrain)
print("Load Train data: ")
dataTrain = pd.read_excel(pathToTrain, sheet_name=0, sep='delimiter',
                           header=None)
dataTrain = dataTrain.values[:Ntrain,:]
labelsTrain = np.repeat(0, Ntrain)

for sheet in range(1,8):
    ms = pd.read_excel(pathToTrain, sheet_name=sheet,
                       sep='delimiter', header=None)
    ms = ms.values[:Ntrain,:]
    dataTrain = np.append(dataTrain, ms)
    labelsTrain = np.append(labelsTrain, np.repeat(sheet, Ntrain))

dataTrain = dataTrain.reshape(Ntrain*8, 6, 4)
print("Example of matrix input: ", dataTrain[-1])
labelsTrain = to_categorical(labelsTrain)
print(labelsTrain)
print("Shape of input: ", dataTrain.shape)
print("Shape of labels: ", labelsTrain.shape)

print("Load test data: ")
Ntest = 40 # Number matrix for 1 disease
dataTest = pd.read_excel(pathToTest, sheet_name=0, sep='delimiter',
                          header=None)
dataTest = dataTest.values[:Ntest,:]
labelsTest = np.repeat(0, Ntest)

for sheet in range(1,8):
    ms = pd.read_excel(pathToTest, sheet_name=sheet,
                       sep='delimiter', header=None)
    ms = ms.values[:Ntest,:]
    dataTest = np.append(dataTest, ms)
    labelsTest = np.append(labelsTest, np.repeat(sheet, Ntest))
```

```

dataTest = dataTest.reshape(Ntest*8, 6, 4)
print("Example of matrix input: ", dataTest[0])
labelsTest = to_categorical(labelsTest)
print(labelsTest)
print("Shape of input: ", dataTest.shape)
print("Shape of labels: ", labelsTest.shape)

model = Sequential()
model.add(Flatten(input_shape=(6,4)))
model.add(Dense(24, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(8, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(dataTrain, labelsTrain, epochs=20, batch_size=20,
verbose=1, validation_split=0.2)
test_loss, test_acc = model.evaluate(dataTest, labelsTest)
print('Acc: ', test_acc)

```