Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра «Системы автоматики, автоматизированное управление
и проектирование»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ С.В. Ченцов

«____» _____ 2021 г.

БАКАЛАВРСКАЯ РАБОТА

15.03.04 – Автоматизация технологических процессов и производств

СИСТЕМА АВТОМАТИЧЕСКОГО РАСПОЗНАВАНИЯ СПЕЦИФИКАЦИИ РЕЗИСТОРА

			доцент, д-р техн. наук
Руководитель	_____	__.06. 2021 г.	Е. Д. Агафонов
Выпускник	_____	__.06. 2021 г.	Д. Д. Санжитов
Нормоконтролер	_____	__.06. 2021 г.	Т.А. Грудинова

Красноярск 2021

РЕФЕРАТ

Выпускная квалификационная работа на тему «Система автоматического распознавания спецификации резистора» содержит 68 страниц текстового документа, 2 приложения, 18 использованных источников.

РЕЗИСТОР, PYTHON, МАШИННОЕ ОБУЧЕНИЕ, ГЛУБОКОЕ ОБУЧЕНИЕ, КОЛОРИСТИКА, МАШИННОЕ ЗРЕНИЕ, ТЕХНИЧЕСКОЕ ЗРЕНИЕ.

Объектом разработки является система автоматического распознавания номинала резистора.

Предметом разработки является мобильная система автоматического распознавания спецификации резистора с помощью технологий машинного зрения.

Цель работы**:** разработка системы автоматического распознавания номинала резистора с помощью машинного зрения с использованием последовательности цветовой маркировки резистора.

Задачи, которые решались в ходе выполнения бакалаврской работы:

1) анализ предметной области;
2) разработка telegram бота;
3) переобучение нейросети для определения резистора;
4) разработка алгоритма обрезки резистора;
5) разработка алгоритма выделения цветных полос;
6) разработка алгоритма определения цвета;
7) разработка алгоритма определения номинала резистора.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Основную часть информации человек получает по зрительному каналу, и далее полученную информацию использует для решения различных задач. Множество трудоемких задач решаются с помощью визуальной информации, и поэтому встает вопрос автоматизации решения данных задач с помощью машинного зрения.

Машинное зрение — это научное направление в области искусственного интеллекта, в частности робототехники, и связанные с ним технологии получения изображений объектов реального мира, их обработки и использования полученных данных для решения разного рода прикладных задач без участия (полного или частичного) человека [1].

Машинное зрение решает следующие задачи:

- идентификация;
- обнаружение;
- распознавание текста;
- восстановление сцены;
- восстановление изображений;
- выделение на изображениях структур определенного вида, сегментация изображений;
- анализ оптического потока;
- восстановление 3D формы по 2D изображениям;
- оценка движения.

Технологии машинного зрения используются в многих востребованных областях науки и техники, таких как автоматизация процессов, военные технологии, оптимизация технологического процесса, повышение качества выпускаемых изделий, контроль производственного оборудования, интеллектуальные робототехнические комплексы, системы управления движущимися аппаратами, и множество других. Также технология машинного

зрения требуется в сфере проверки печатных плат, диагностики и ремонта различной электроники.

Целью выпускной квалификационной работы является разработка системы автоматического распознавания номинала резистора с помощью машинного зрения используя последовательность цветной маркировки.

Для выполнения цели были поставлены следующие задачи:

1) анализ предметной области;
2) разработка telegram бота;
3) переобучение нейросети для определения резистора;
4) разработка алгоритма обрезки резистора;
5) разработка алгоритма выделения цветных полос;
6) разработка алгоритма определения цвета;
7) разработка алгоритма определения номинала резистора.

Объектом разработки является система автоматического распознавания спецификации резистора.

Предметом разработки является мобильная система автоматического распознавания спецификации резистора с помощью технологий машинного зрения.

1 Система автоматического распознавания спецификации резистора

1.1 Общая информация о резисторах

1.1.1 Общая характеристика резисторов

Резисторы применяются практически во всех видах радиоэлектронной аппаратуры для регулирования и распределения электрической энергии.

Когда-то резисторы назывались сопротивлениями, но в соответствии с Государственным стандартом [2] электрическим сопротивлениям, как схемным элементам, присвоено название «резисторы».

Сделано это было с целью различать «сопротивление» как изделие (радиокомпонент) и «сопротивление», как его физическое свойство, электрическую величину. Резисторы характеризуются электрическим сопротивлением.

Основной единицей электрического сопротивления в соответствии с международной системой единиц является Ом [3]. На практике используются также производные единицы — килоом (кОм), мегаом (МОм), гигаом (ГОм), тераом (ТОм), которые связаны с основной единицей следующими соотношениями:

– $1\ \text{кОм} = 10^3\text{Ом}$;

– $1\ \text{МОм} = 10^6\text{Ом}$;

– $1\ \text{ГОм} = 10^9\text{Ом}$;

– $1\ \text{ТОм} = 10^{12}\text{Ом}$.

Резисторы бывают следующих видов: постоянные и переменные [4]. Переменные также делятся на регулировочные и подстроечные. У постоянных резисторов сопротивление не изменяется в процессе эксплуатации.

Резисторы, которые осуществляют регулировку аппаратуры путем изменения сопротивления резистора, называются потенциометрами или

переменными резисторами. Резисторы, изменяющиеся только в процессе налаживания радиоэлектронной аппаратуры, называются подстроечными.

1.1.2 Основные параметры резисторов

Резисторы характеризуются следующими основными параметрами: номинальным значением сопротивления, допустимым отклонением сопротивления от номинального значения, номинальной мощностью рассеяния, максимальным рабочим напряжением, температурным коэффициентом сопротивления, собственными шумами и коэффициентом напряжения.

Номинальное значение сопротивления R обычно обозначено на корпусе резистора. Действительное сопротивление может быть отклоняться от номинального сопротивления в пределах допустимого отклонения.

1.2 Маркировка резисторов

1.2.1 Цифробуквенная маркировка резисторов

Резисторы времен СССР были с цифробуквенной маркировкой, с помощью которых можно было определить характеристики данного элемента [2]. Они состояли из двух или трех цифр и латинских или русских букв. Цифры обозначали значащие цифры номинала резистора, а буква являлась множителем или допуском.

На данный момент цифробуквенная маркировка резисторов не применяется, но элементная база до сих пор существует в различной старой советской электронике и еще работоспособна.

Таблица 1.1 - Расшифровка буквенных обозначений в маркировке резисторов старого образца

Сопротивление		Допуск	
Множитель	Код	Допуск, %	Код
1	R (Е)	± 0.1	B (Ж)
		± 0.23	C (У)
10^3	K (К)	± 0.5	D (Д)
		± 1	F (Р)
10^6	M (М)	± 2	G (Л)
		± 5	J (И)
10^9	G (Г)	± 10	K (С)
		± 20	M (В)
10^{12}	T (Т)	± 30	N (Ф)

Приведенная таблица позволяет идентифицировать номинал резистора с цифробуквенной маркировкой. Например, резистор с марикровкой 210RJ будет означать сопротивление 210 Ом с допуском 5%. Если маркировка состоит только из цифр, то данный резистор обозначается в Омах, а допуск равен 20%.

1.2.2 Цветовая маркировка резисторов

Цветовая маркировка резисторов — это набор цветных полос на корпусе элемента, каждому из которых соответствует определенный цифровой код.

В соответствии с гостом и требованиям Международной Электротехнической Комиссии (IEC) маркировка наносится в виде четырех, пяти или шести цветовых полос [5]. Первая полоса находится ближе к краю.

Таблица 1.2 - Количество полос и значение их местоположения

	3 полосы	4 полосы	5 полос	6 полос
1 полоса	1-я цифра	1-я цифра	1-я цифра	1-я цифра
2 полоса	2-я цифра	2-я цифра	2-я цифра	2-я цифра
3 полоса	Множитель	Множитель	3-я цифра	3-я цифра
4 полоса	-	Погрешность	Множитель	Множитель
5 полоса	-	-	Погрешность	Погрешность
6 полоса	-	-	-	Температурный коэффициент

Таблица 1.3 – Соответствие цвета полосы и значения их местоположения

Цвет\Значение	Значащие цифры	Множитель	Погрешность [%]	ТКС $[\frac{‰}{°C}]$
Черный	0	1		
Коричневый	1	10	± 1	100
Красный	2	100	± 2	50
Оранжевый	3	1000		15
Желтый	4	10^4		25
Зеленый	5	10^5	± 0.5	
Синий	6	10^6	± 0.25	10
Фиолетовый	7	10^7	± 0.1	5
Серый	8	10^8	± 0.05	
Белый	9	10^9		1
Золотистый		10^{-1}	± 5	
Серебристый		10^{-2}	± 10	

С помощью таблиц 1.2 и 1.3 можно определить значение номинала резистора. Например, резистор с желтым, красным, зеленым, фиолетовым, серым полосами в соответствующем порядке будет с номиналом 4.25 ГОм и точностью 0.05%.

1.2 Аналоги автоматического распознавания номинала резистора

Проанализировав рынок решений, нашлось два продукта с автоматическим распознаванием номинала путем анализа изображения. Первый продукт — это Resistor Scanner от разработчика MhTechDev. Приложение представляет собой приложение на операционной системе Android, с возможностью автоматически распознавать номинал резистора с помощью фото.

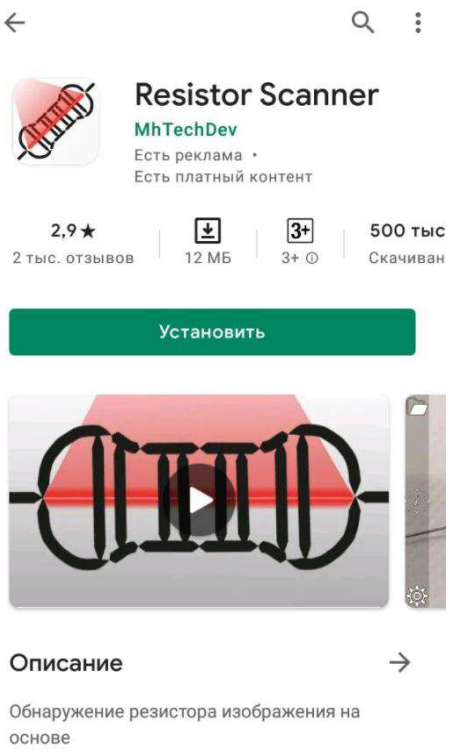

Рисунок 1.1 – Страница продукта в магазине Play Маркет

Для определения номинала требуется сначала выбрать количество полос на резисторе в главном меню и произвести фотосъемку. Количество полос может быть также в автоматически настраиваемом режиме.

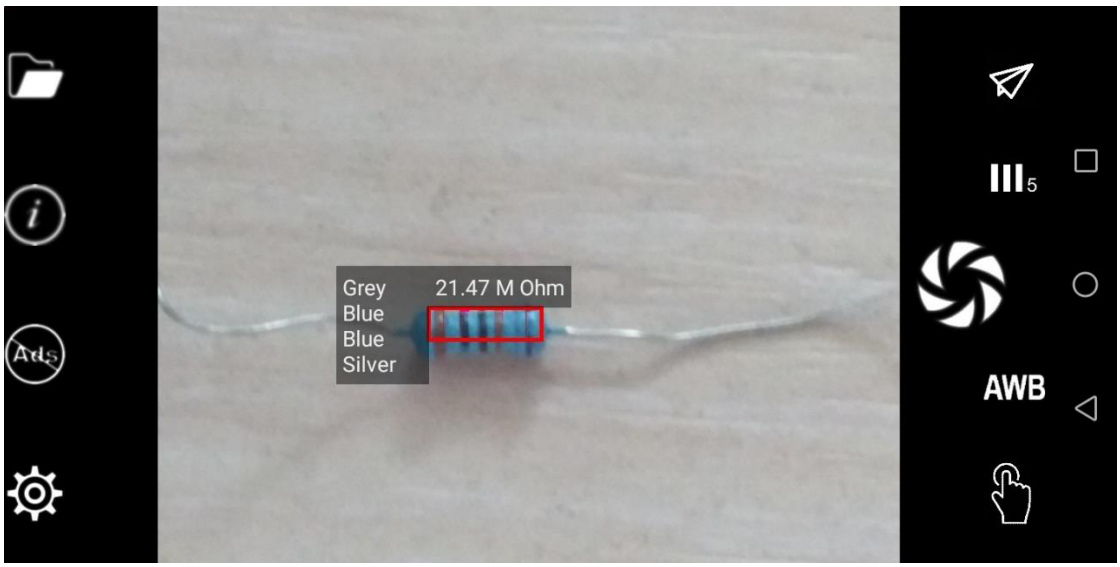

Рисунок 1.2 – Скриншот работы программы

В случае неправильного определения цвета полос, можно произвести ручную регулировку.

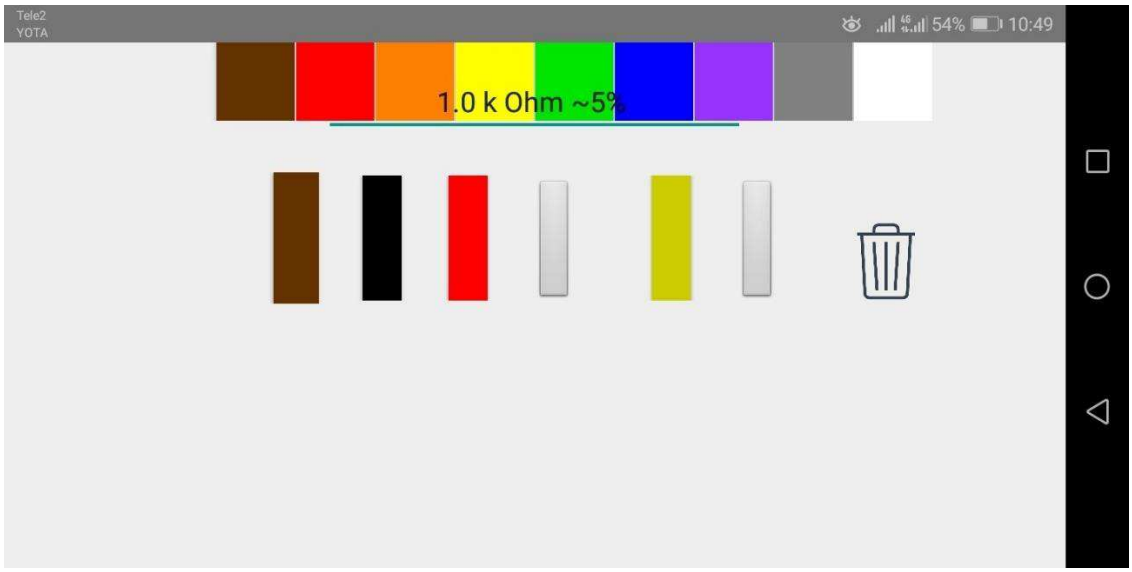

Рисунок 1.3 – Скриншот работы программы

Также данная программа позволяет загружать изображения с галереи телефона, настраивать фокусное расстояние и баланс белого.

Следующим продуктом является программа Resistor Scaner(beta) от разработчика Иoldy Labs.

Рисунок 1.4 – Страница продукта в магазине Play Маркет

Для определения номинала резистора требуется сделать фотосъемку резистора в центре ограничительных рамок программы.

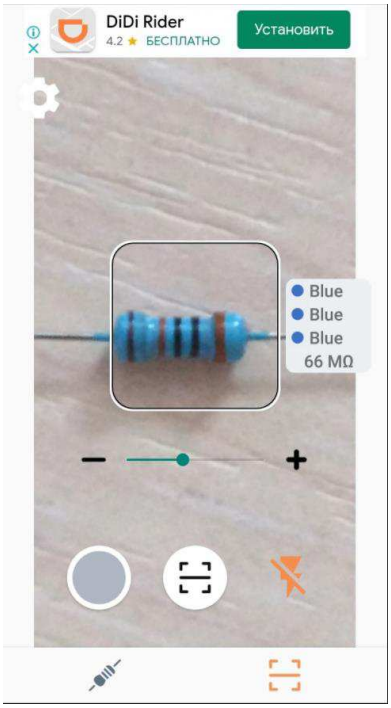

Рисунок 1.5 – Скриншот работы программы

Программа работает в двух режимах: обычный и live. Обычный режим позволяет сделать фото и показывает результат на фиксированном изображении. В live режиме результат работы накладывается прямо на работу камеры. Также данная программа позволяет настраивать фокусное расстояние и работу вспышки камеры.

2 Алгоритмическая и программная реализация интеллектуальной системы распознавания спецификации резистора на мобильной платформе

Для создания нового решения потребуются способы определения местоположения резистора, выделения положения полос и корректного распознавания цвета полос резистора.

2.1 Определение местоположения фото

Для определения местоположения резистора можно воспользоваться нейронными сетями.

2.1.1 Нейронные сети

Нейронная сеть – это математическая модель в виде программного и аппаратного воплощения, строящаяся на принципах функционирования биологических нейросетей [6].

Работа с изображениями является одной из важных сфер с использованием технологий Нейронных сетей. Нейронная сеть для распознавания образов – это самый популярный способ применения нейронных сетей.

В качестве распознаваемых образов могут выступать различные объекты. Например, нейронные сети используют для распознавания текста или для распознавания животных на изображении. При обучении сети предлагаются

различные изображения с наличием образов для распознавания и метаданные, с указанием к какому классу относится тот или иной образ.

Выделяются несколько следующих архитектур [7] для распознавания изображений:

- многослойный перцептрон;
- сверточная;
- рекурсивная;
- рекуррентная;
- сеть долгой краткосрочной памяти;
- sequence-to-sequence модель
- неглубокие

Существует несколько способов обучить нейронную сеть:

- машинное обучение с учителем;
- машинное обучение без учителя;
- машинное обучение с частичным привлечением учителя;
- машинное обучение с подкреплением.

Также нейронные сети используют для принятия решений и управления, аппроксимации, кластеризации, прогнозировании, сжатия данных, ассоциативной памяти, анализа данных и оптимизации.

Для определения местоположения резистора переобучим готовую нейронную сеть для определения образов.

При переобучении воспользуемся Tensorflow Object Detection API [8], открытой программной библиотекой для машинного обучения, разработанной компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия.

2.1.2 Подготовка данных

В первую очередь требуется собрать набор данных, достаточных для обучения.

Выбор данных для обучения сети и их обработка является самым сложным этапом решения задачи. Чтобы собрать правильный набор данных, следует поддерживаться нескольких критериев:

– Репрезентативность — данные должны иллюстрировать истинное положение вещей в предметной области;

– Непротиворечивость — противоречивые данные в обучающей выборке приведут к плохому качеству обучения сети.

На рисунке 2.1 представлены различные фотографии с наличием резисторов на разном фоне и с различным освещением, которые войдут в обучаемый набор.

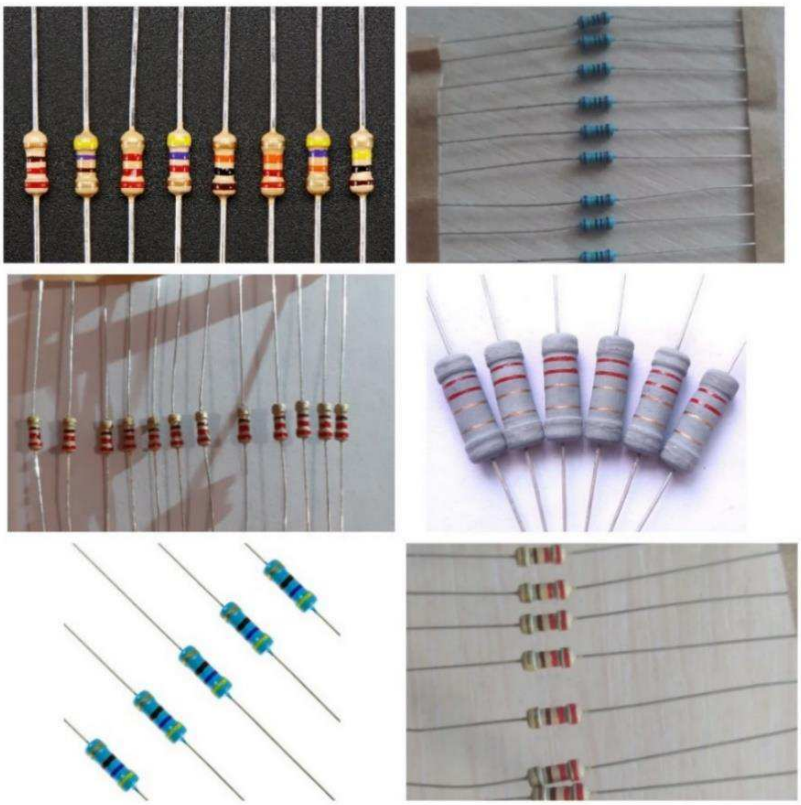

Рисунок 2.1 – Пример фотографий для обучения модели

Для обучения было собрано 500 фотографий с размерами 800x600px. На фотографии в среднем 6 резисторов, что составляет примерно 3000 экземпляров нужного объекта.

Следующим этапов обучения является создание разметки. Для этого воспользуемся open source проектом на python «LabelImg».

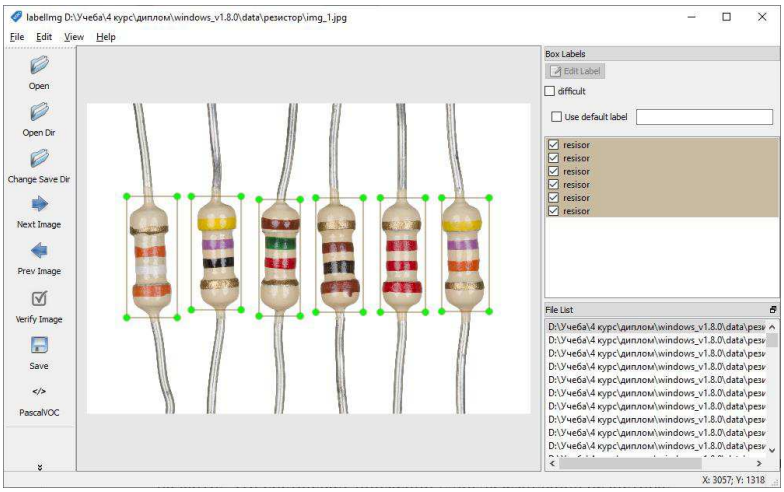

Рисунок 2.2 – Скриншот работы программы LabelImg

Программа позволяет производить разметку в виде прямоугольников с нужными объектами и сохранять данные в типе PascalVOC(*.xml).

Далее скачиваем готовую модель для переобучения. Это будет модель ssdlite_mobilenet_v2_coco.

Mobile Net V2 является сетью для мобильного визуального распознавания, включая классификацию, обнаружение объектов и семантическую сегментацию. Mobile Net V2 выпускается как часть библиотеки классификации изображений TensorFlow-Slim,

Мобильная сеть V2 строится на идеях мобильной сети V1, используя разделяемую свертку по глубине в качестве эффективных строительных блоков. Однако версия V2 вводит в архитектуру две новые функции:

1) линейные узкие места между слоями;

2) быстрые соединения между узкими местами.

2.1.3 Обучение модели

Для обучения воспользуемся Google Collaboratory [9], бесплатной интерактивной облачной средой для работы с кодом от Google. Данный сервис предлагает бесплатно воспользоваться GPU уровня Tesla K80.

Загрузим данные в Google Drive и скачаем готовый блокнот object_detection.ipynb.

Обучение состоит из следующих этапов:

1) клонирование репозитория TensorFlow Models;
2) установка protobuf и компиляция необходимых файлов;
3) добавление необходимых путей в PYTHONPATH;
4) установка PyDrive для получения файлов с Google Drive;
5) скачивание архива и его реархивация;
6) запуск процесса обучения;
7) конвертация результата обучения в frozen graph;
8) конвертация обученной модели в tflite.

Также протестируем модель при помощи нескольких фотографий. Пример работы модели представлен на рисунке 2.3.

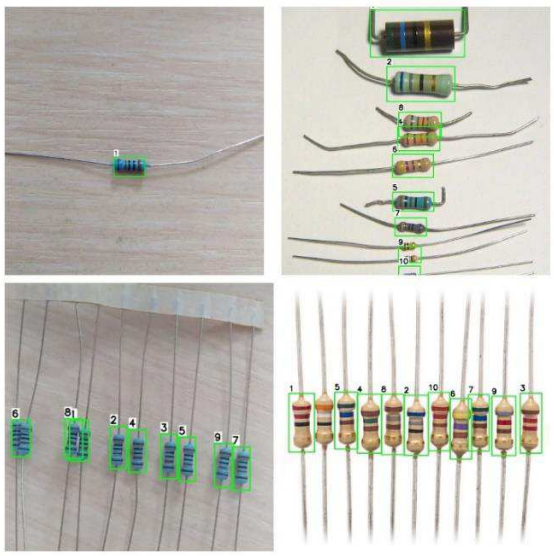

Рисунок 2.3 – Пример работы модели

На рисунке видим, что модель работает корректно. Структура переобученной модели представлена в приложении А.

С помощью данной нейронной сети обрезаем каждый резистор и сохраняем в отдельный файл.

Рисунок 2.4 – Пример обрезанного резистора с помощью нейросети

Скрипт, который используется для выделения резистора с фотографии находится в приложении Б.

2.2 Определение границ резистора

Обученная нейронная сеть определяет номинал резистора с малым количеством излишнего пространства. Данное пространство препятствует корректному определению номинала резистора.

Рисунок 2.5 – Пример излишнего пространства

Из рисунка 2.5 видно, что изображение резистора состоит из самого резистора и излишнего пространства вокруг резистора, которое находится вне красного прямоугольника.

Определение границ резистора проходит в несколько этапов:

1) Получение изображения;

2) Использование детектора границ Canny.

Детектор границ Canny является фильтром изображения, который находит кривые, вдоль которых происходит резкое изменение яркости или других видов неоднородностей.

Шаги детектора:

- удаление шума и лишних деталей из изображения;
- расчет градиента изображения;
- сужение краев;
- связка краев.

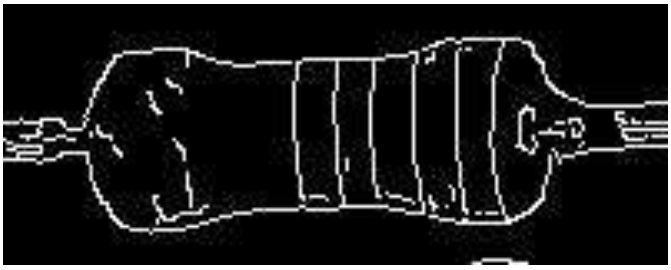

Рисунок 2.6 – Изображение с фильтром детектора Canny

3) Создание маски с помощью минимальной выпуклой оболочки и обрезка по маске

Минимальная выпуклая оболочка — это такая оболочка, которая является выпуклым многоугольником наименьшего периметра.

Рисунок 2.7 – Разновидности оболочек

Изображение, полученное в пункте 2, проведем через алгоритм нахождения минимальной выпуклой оболочки.

Рисунок 2.8 – Пример маски МВО для резистора

Рисунок 2.9 – Изображение после обрезки по маске

4) Обрезка излишеств по краям резистора

Из рисунка 2.9 можно сделать вывод, что изображение содержит лишние части слева и справа от резистора.

С помощью функции getColor из библиотеки ColorThief определяем самый распространённый цвет и принимаем его за цвет корпуса резистора. Далее рассматриваем каждый столбец изображения и, когда количество пикселей в столбце равных цвету пикселя превысит 35% высоты изображения, то это будет означать границу изображения.

Рисунок 2.10 – Финальное изображение

Таким образом, получаем финальное изображение, которое состоит только из самого резистора.

2.3 Корректное определение цвета пикселя

2.3.1 Цветовое пространство RGB

RGB (red, green, blue — красный, зелёный, синий) — аддитивная цветовая модель, описывающая способ кодирования цвета для цветовоспроизведения с помощью трёх цветов, которые принято называть основными. Выбор основных цветов обусловлен особенностями физиологии восприятия цвета сетчаткой человеческого глаза [10].

RGB-модель является аддитивной, где цвета получаются путём добавления к чёрному цвету. При отсутствии излучения — нет никакого цвета — чёрный, смешение всех трёх в определённой пропорции — даёт белый [11].

В первых телевизорах и мониторах применялись три электронных пушки для красного, зелёного и синего каналов. В ЖК- и других матричных мониторах и телевизорах для смешения трёх цветов помогают светоточки.

Для большинства приложений значения координат r, g и b можно считать принадлежащими отрезку [0,1], что представляет пространство RGB в виде куба 1×1×1, который представлен на рисунке 2.11.

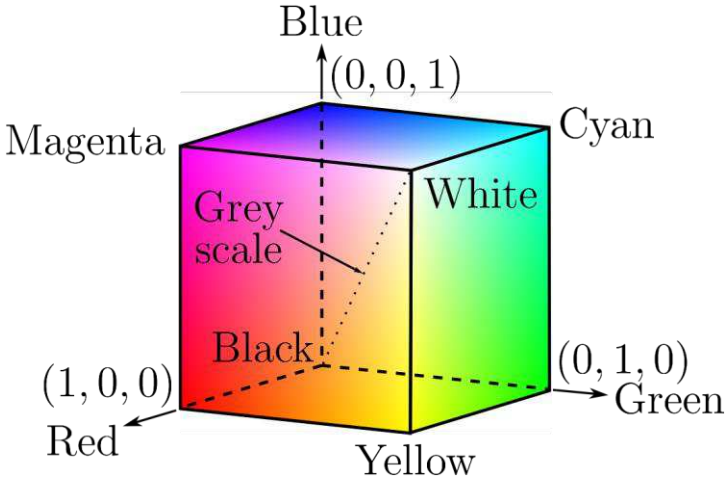

Рисунок 2.11 – RGB куб

Цветные изображения в модели RGB строятся из трёх отдельных изображений-каналов, которые можно увидеть на рисунке 2.12.

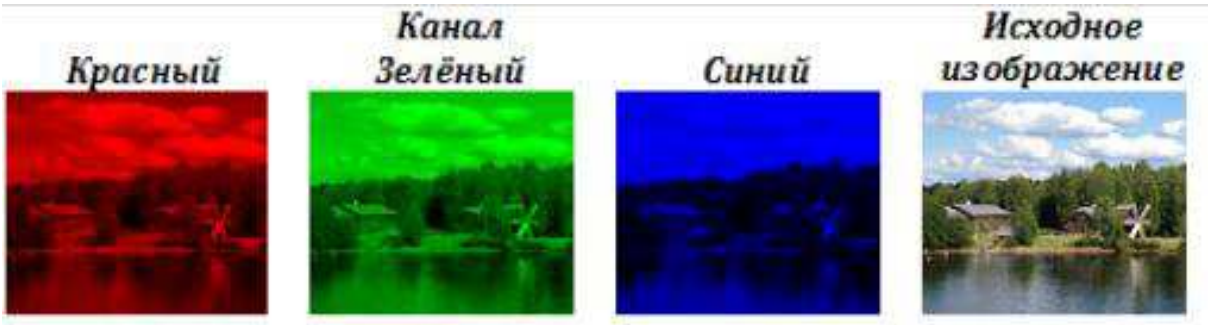

Рисунок 2.12 – Разложение исходного изображения на 3 канала

В ЭВМ для представления каждой из координат представляются в виде одного октета, значения которого обозначаются для удобства целыми числами от 0 до 255, где 0 — минимальная интенсивность, а 255 — максимальная интенсивность.

В предлагаемом решении все пиксели считываются в RGB пространстве и конвертируются в другие.

2.3.2 Цветовое пространство CIE XYZ

С целью унификации была разработана международная стандартная цветовая модель CIE XYZ[12]. В результате серии экспериментов международная комиссия по освещению (CIE) определила кривые сложения основных (красного, зелёного и синего) цветов. В этой системе каждому видимому цвету соответствует определённое соотношение основных цветов. Для отображения всех видимых цветов пришлось ввести отрицательное количество базовых цветов. Чтобы уйти от отрицательных значений, CIE ввела нереальные или мнимые основные цвета: X (мнимый красный), Y (мнимый зелёный), Z (мнимый синий).

При описании цвета значения X, Y, Z называют стандартными основными возбуждениями, а полученные на их основе координаты – стандартными цветовыми координатами.

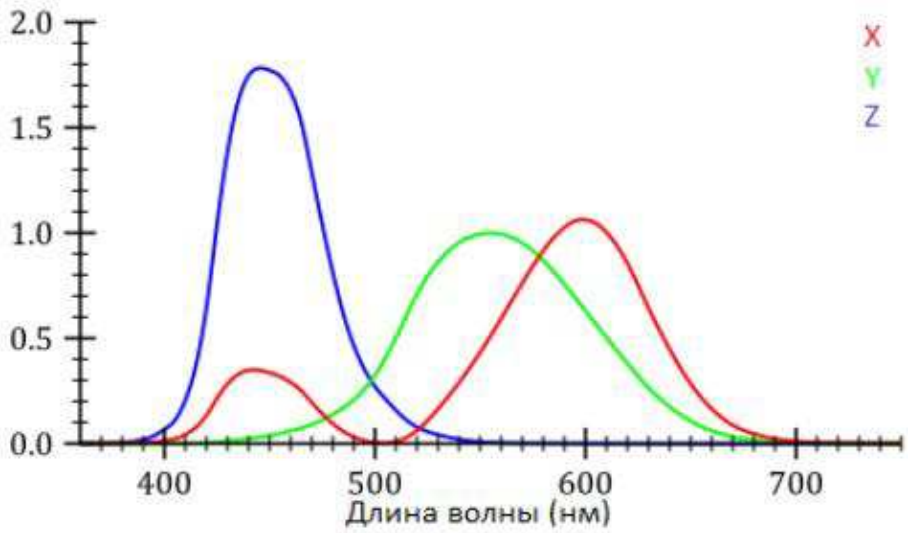

Рисунок 2.13 – Кривые сложения X(λ), Y(λ), Z(λ)

Помимо стандартных цветовых координат часто используют понятие относительных цветовых координат, которые можно вычислить по следующим формулам:

$$\begin{cases} X = \frac{X}{X+Y+Z} \\ Y = \frac{Y}{X+Y+Z} \\ Z = \frac{Z}{X+Y+Z} \end{cases} \quad (2.1)$$

Легко заметить, что X+Y+Z=1, что значит для однозначного задания относительных координат достаточно любой пары значений, а соответствующее цветовое пространство может быть представлено в виде двумерного графика:

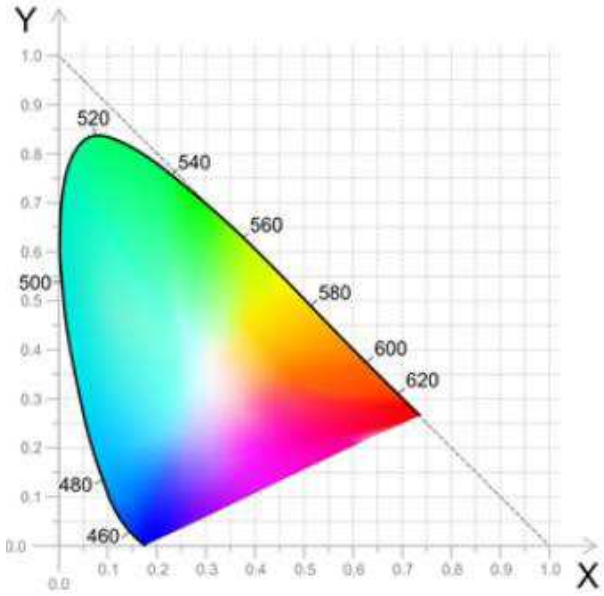

Рисунок 2.14 – Двумерный график цветового пространства

Множество цветов, задаваемое таким способом, называют треугольником CIE. Легко заметить, что треугольник CIE описывает только цветовой тон, но никак не описывает яркость. Для описания яркости вводят дополнительную ось, проходящую через точку с координатами (0.33;0.33).

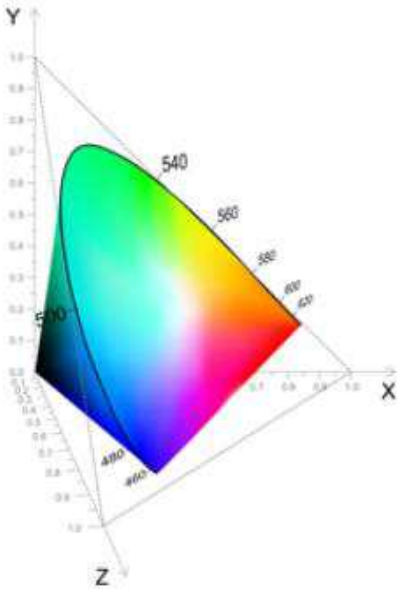

Рисунок 2.15 – Цветовое тело CIE XYZ

Это тело содержит все цвета, видимые среднестатистическому наблюдателю. Основным недостатком этой системы является то, что, используя её, можно констатировать только совпадение или различие двух цветов, но расстояние между двумя точками этого цветового пространства не соответствует зрительному восприятию различия цветов.

2.3.3 Цветовое пространство CIE LAB

Основной целью при разработке CIE LAB [13] было устранение нелинейности системы CIE XYZ с точки зрения человеческого восприятия. Под аббревиатурой LAB обычно понимается цветовое пространство CIE L*a*b*, которое на данный момент является стандартом, принятым международной комиссией по освещению.

В системе CIE L*a*b координата L означает светлоту (в диапазоне от 0 до 100), координаты a* обозначает позицию между зелёным и пурпурным, а b* показывает позицию между синим и жёлтым цветами.

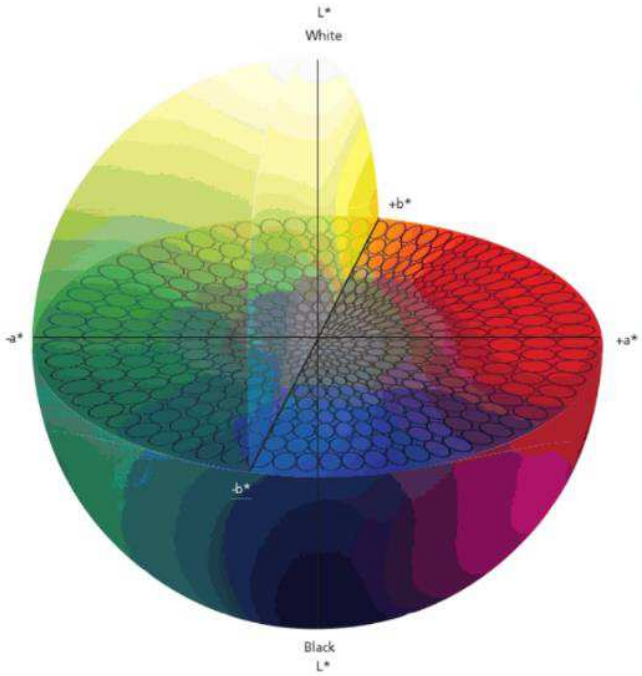

Рисунок 2.16 – Цветовое тело CIE LAB

2.3.4 Перевод из цветового пространства RGB в цветовое пространство CIE LAB

Конвертация из цветового пространства RGB в цветовое пространство CIE lab проходит в 2 этапа:

1) Конвертация из RGB в CIE XYZ [14].

$$X = 0.418 * R - \ 0.091 * G + 0.001 * B \quad (2.2)$$

$$Y = -0.158 * R + \ 0.252 * G - 0.003 * B \quad (2.3)$$

$$Z = -0.083 * R + \ 0.016 * G - 0.179 * B \quad (2.4)$$

2) Конвертация из CIE XYZ в CIE LAB [14].

$$L = 116 f_y - 16 \quad (2.5)$$

$$a = 500\left(f_x - f_y\right) \tag{2.6}$$

$$b = 200\left(f_y - f_z\right) \tag{2.7}$$

$$f_x = \begin{cases} \sqrt[3]{x_r}, \text{если } x_r > \in \\ \dfrac{kx_r + 16}{116}, \text{иначе} \end{cases} \tag{2.8}$$

$$f_x = \begin{cases} \sqrt[3]{x_r}, \text{если } x_r > \in \\ \dfrac{kx_r + 16}{116}, \text{иначе} \end{cases} \tag{2.9}$$

$$f_x = \begin{cases} \sqrt[3]{x_r}, \text{если } x_r > \in \\ \dfrac{kx_r + 16}{116}, \text{иначе} \end{cases} \tag{2.10}$$

$$x_r = \frac{X}{X_r} \tag{2.11}$$

$$y_r = \frac{Y}{Y_r} \tag{2.12}$$

$$z_r = \frac{Z}{Z_r} \tag{2.13}$$

2.3.5 Формула цветового отличия

Формула цветового отличия, цветоразность, или цветовое расстояние - математическое представление, позволяющее численно выразить различие между двумя цветами в колориметрии [15]. Распространенные определения цветового различия обычно используют формулу вычисления расстояния в

евклидовом пространстве, однако стоит заметить, что при этом не каждое цветовое пространство является евклидовым со строгой математической точки зрения.

Международный комитет CIE задает определение цветового отличия через метрику ΔE.

2.3.5.1 CIE76

Цветовое отличие можно найти, используя координаты (L_1^*, a_1^*, b_1^*) и (L_2^*, a_2^*, b_2^*) в цветовом пространстве L*a*b*:

$$\Delta E_{ab}^* = \sqrt{(L_2^* - L_1^*)^2 + (a_2^* - a_1^*)^2 + (b_2^* - b_1^*)^2} \quad (2.14)$$

2.3.5.2 CIE94

ΔE (1994) задавалось в цветовом пространстве LCH (L*C*h).

$$\Delta E_{ab}^* = \sqrt{\left(\frac{L_2^* - L_1^*}{K_L}\right)^2 + \left(\frac{C_2^* - C_1^*}{1 + K_1 C_1^*}\right)^2 + \left(\frac{h_2 - h_1}{1 + K_2 C_1^*}\right)^2} \quad (2.15)$$

Таблица 2.1 – значения коэффициента K в зависимости от области применения

	Искусство	Промышленность
K_L	1	2
K_1	0.045	0.048
K_2	0.015	0.014

2.3.5.3 CIEDE2000

Определение 1994 года не полностью устранило неоднородности восприятия цветового различия, и поэтому комитет CIE разработал новый стандарт [16], который включал пять дополнений:

– поворот цветового угла тона (R_T), чтобы устранить проблемы в синей области (угол Hue 275°);

– компенсация для нейтральных цветов;

– компенсация для светлоты (S_L);

– компенсация для насыщенности цвета (S_C);

– компенсация для тона (S_H).

$$\Delta E_{ab}^{*} = \sqrt{\left(\frac{\Delta L'}{S_L}\right)^2 + \left(\frac{\Delta C'}{S_c}\right)^2 + \left(\frac{\Delta H'}{S_H}\right)^2 + R_T \frac{\Delta C'}{S_c}\frac{\Delta H'}{S_H}} \quad (2.16)$$

$$\bar{L} = \frac{L_1^* + L_2^*}{2} \quad (2.17)$$

$$\bar{C} = \frac{C_1^* + C_2^*}{2} \quad (2.18)$$

$$a_1' = a_1 + \frac{a_1}{2}\left(1 - \frac{1}{2}\sqrt{\frac{\bar{C}^7}{C^7 + 25^7}}\right) \quad (2.19)$$

$$a_2' = a_2 + \frac{a_2}{2}\left(1 - \frac{1}{2}\sqrt{\frac{\bar{C}^7}{C^7 + 25^7}}\right) \quad (2.20)$$

$$b_1' = b_1 + \frac{b_1}{2}\left(1 - \frac{1}{2}\sqrt{\frac{\bar{C}^7}{C^7 + 25^7}}\right) \quad (2.21)$$

$$b_2' = b_2 + \frac{b_2}{2}\left(1 - \frac{1}{2}\sqrt{\frac{\bar{C}^7}{C^7 + 25^7}}\right) \quad (2.22)$$

$$\bar{C} = \frac{C_1' + C_2'}{2} \tag{2.23}$$

$$\Delta C' = C_1' - C_2' \tag{2.24}$$

$$\mathrm{C}_1' = \sqrt{a_1'^2 + b_1'^2} \tag{2.25}$$

$$\mathrm{C}_2' = \sqrt{a_2'^2 + b_2'^2} \tag{2.26}$$

$$h_1' = arctg\left(\frac{b_1}{a_1'}\right) \quad mod\ 2\pi \tag{2.27}$$

$$h_2' = arctg\left(\frac{b_2}{a_2'}\right) \quad mod\ 2\pi \tag{2.28}$$

$$\Delta h' = \begin{cases} h_2' - h_1' & |h_1' - h_2'| \le \pi \\ h_2' - h_1' + 2\pi & |h_1' - h_2'| > \pi, h_2' \le h_1' \\ h_2' - h_1' - 2\pi & |h_1' - h_2'| > \pi, h_2' > h_1' \end{cases} \tag{2.29}$$

$$\Delta H' = 2\sqrt{C_1' C_2'} \sin\left(\frac{\Delta h'}{2}\right) \tag{2.30}$$

$$\bar{H}' = \begin{cases} \dfrac{(h_1' - h_2' + 2\pi)}{2} & |h_1' - h_2'| > \pi \\ \dfrac{(h_1' - h_2')}{2} & |h_1' - h_2'| \le \pi \end{cases} \tag{2.31}$$

$$T = 1 - 0.17\ \cos\left(\bar{H}' - \frac{\pi}{6}\right) + 0.24\cos(2\bar{H}') + 0.32\cos\left(3\bar{H}' + \frac{\pi}{30}\right) - \\ - 0.20\cos\left(4\bar{H}' - \frac{\pi}{20}\right) \quad (2.32)$$

$$S_L = 1 + \frac{0.015(\bar{L} - 50)^2}{\sqrt{20 + (L - 50)^2}} \quad (2.33)$$

$$S_C = 1 + 0.045\bar{C}' \quad (2.34)$$

$$S_H = 1 + 0.15\bar{C}'T \quad (2.35)$$

$$R_T = -1\sqrt{\frac{\bar{C}^7}{\bar{C}^7 + 25^7}}\sin\left[\frac{\pi}{6}\exp\left(-\left[\frac{\bar{H}' - \frac{55\pi}{36}}{\frac{5\pi}{36}}\right]^2\right)\right] \quad (2.36)$$

Используя формулы 2.16 – 2.36, мы можем определить цветовое расстояние между двумя цветами в цветовом пространстве CIE Lab.

В предлагаемом решении, для определения цвета, будем использовать формулы 2.2 – 2.36. Скрипт с использованием данных формул представлен в приложении Б.

2.3.6 Набор цветов

Создадим набор цветов с списком RGB-значений для каждого цвета из коллекции. Каждый цвет будет включать в себя несколько RGB- значений, которые в дальнейшем будут нужны для определения цвета полос резистора.

Таблица 2.2 – Набор цветов с списком RGB-значений

Цвет	RGB-значения
Серебряный	(192, 192, 192)
Золотой	(255, 215, 0), (161, 156, 128)
Черный	(0, 0, 0), (16, 57, 83), (58, 58, 69)
Коричневый	(150, 75, 0), (96, 75, 74), (69, 81, 88), (64, 54, 52), (120, 67, 58), (109, 59, 62)
Красный	(255, 255, 0), (112, 57, 54)
Оранжевый	(255, 165, 0), (243, 73, 21)
Желтый	(255, 255, 0)
Зеленый	(0, 128, 0), (49, 89, 56)
Фиолетовый	(135, 0, 255)
Серый	(128, 128, 128), (120, 118, 124)
Белый	(255, 255, 255), (226, 214, 210)
Синий	(68, 75, 132), (29, 42, 89), (25, 61, 107)

Для определения цвета пикселя преобразуем каждое RGB-значение из таблицы 2.2 в цветовое пространство CIE Lab, и с помощью формул цветового отличия найдем цветовое расстояние между цветом из таблицы и цветом определяемого пикселя. Цвет из набора с минимальным цветовым расстоянием будет корректным.

2.4 Выделение полос резистора

Используя функцию getcolor из библиотеки Colorthief [17] определим самый распространенный цвет на изображении.

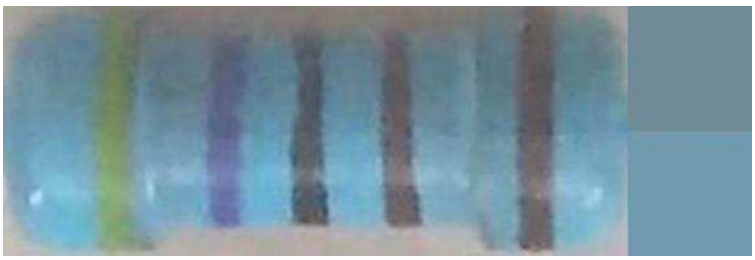

Рисунок 2.17 – Самый распространенный цвет на изображении

За цвет корпуса примем самый распространенный цвет и 2 пикселя из левой центральной стороны.

Осуществим перебор каждого пикселя, находящегося в середине высоты изображения. Используя значения найденного пикселя, цвета корпуса и формулы цветового отличия запомним пиксели с цветом корпуса.

На рисунке 2.18 можно наблюдать линию белого цвета, которая проходит только по цвету корпуса резистора.

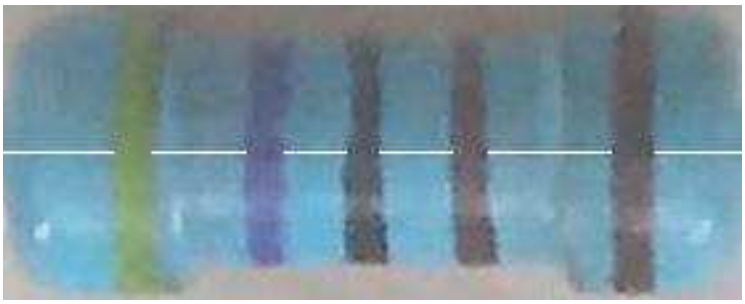

Рисунок 2.18 – Резистор с выделенным корпусом

Таким образом, все пробелы белой полосы можем считать за полосы резистора.

2.5 Сборка в единый проект

Для разработки системы автоматического распознавания спецификации резистора будем использовать язык программирования Python 3.8. Диаграмма структуры проекта изображена на рисунке.

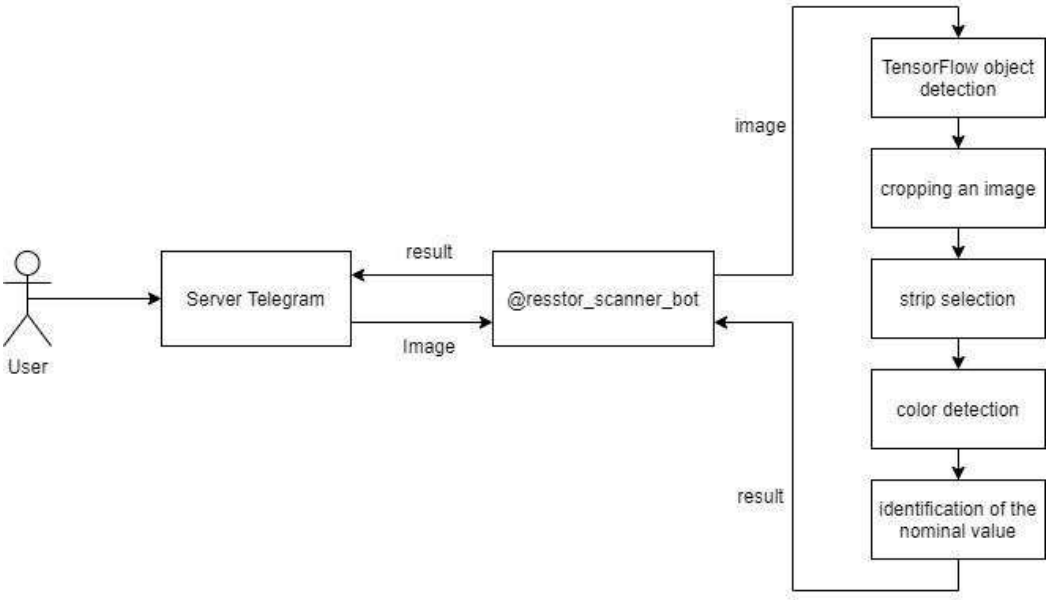

Рисунок 2.19 – Структура проекта

Пользователь отправляет фотографию с резисторами, боту в социальной сети telegram. Бот передает изображение на сервер. Изображение проходит через нейронную сеть и определяет координаты каждого резистора. Также бот рисует прямоугольники на каждом резисторе и отправляет данное изображение пользователю.

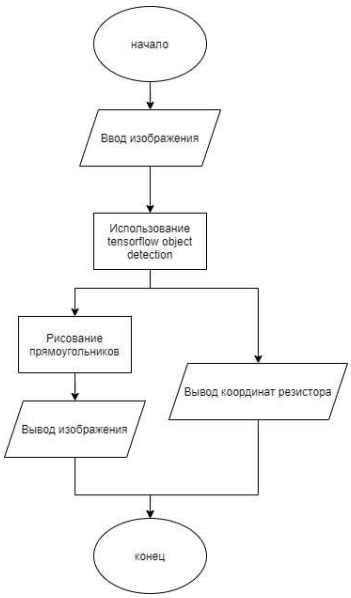

Рисунок 2.20 – Алгоритм определения местоположения резистора

Далее с помощью данных координат обрезается каждый резистор и проводится поиск точных границ корпуса, которые сохраняются в новый файл.

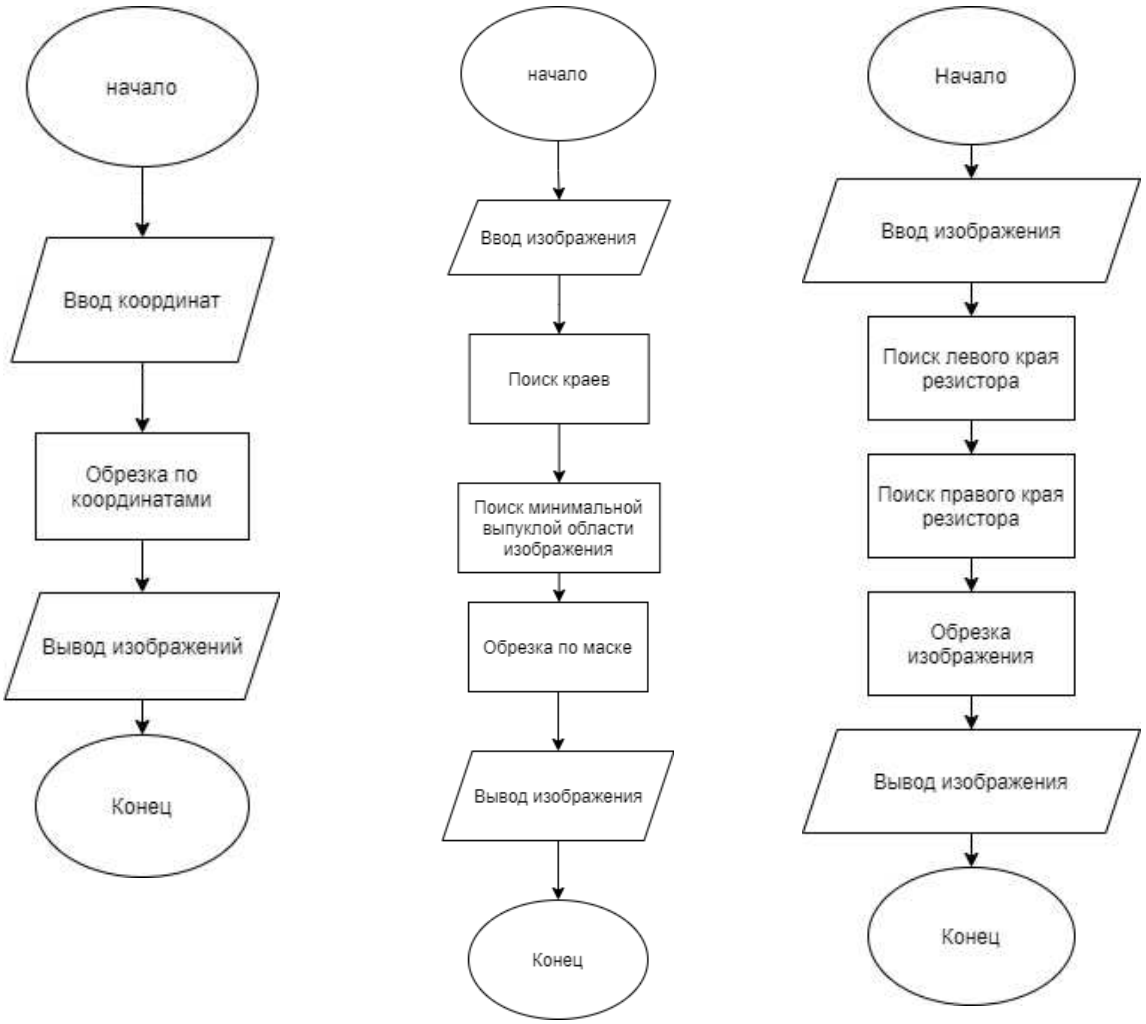

Рисунок 2.21 – Алгоритмы поиска точных границ резистора на изображении

Затем следует поиск координат полос резистора. Найдя координаты полос резистора, получаем RGB значения каждой полосы резистора. Полученные значения переводим в CIE LAB пространство и с помощью формулы цветового отличия находим цвет полосы резистора. Получив последовательность цветов, с помощью таблицы 2 и 3 идентифицировать номинал резистора и отправить его пользователю.

Рисунок 2.22 – Алгоритм определения номинала резистора

Весь код написан на языке python 3.8. Проект состоит из 13 файлов. Описание каждого файла находится в таблице 6.

Таблица 2.3 – Описание файлов в проекте

Название файла	Описание файла
main.py	Telegram бот
TFLite_detection_image.py	Поиск координат каждого резистора и сохранение в отдельный файл
crop1.py	Обрезка изображения по координатам резисторов

Окончание таблицы 2.3

crop2.py	Обрезка по маске, полученной с помощью скрипта convex.py
crop3.py	Обрезка отростков резистора по бокам
edges.py	Поиск краев резистора с помощью оператора Canny
convex.py	Поиск минимальной выпуклой оболочки на изображении
CalculatingTheValue.py	Определение номинала резистора
Color_search.py	Определение минимального цветового расстояния
dominant_color.py	Поиск самого доминантного цвета
rgb_lab.py	Конвертация из rgb в cielab
Search.py	Определение цвета резистора полос и определение номинала
Strip.py	Поиск полос резистора

3 Тестирование системы распознавания спецификации резистора

3.3 Тест работы решения

Для тестирования системы возьмем 10 различных фотографий с наличием резисторов. Фотографии различаются освещением, фоном и количеством резисторов.

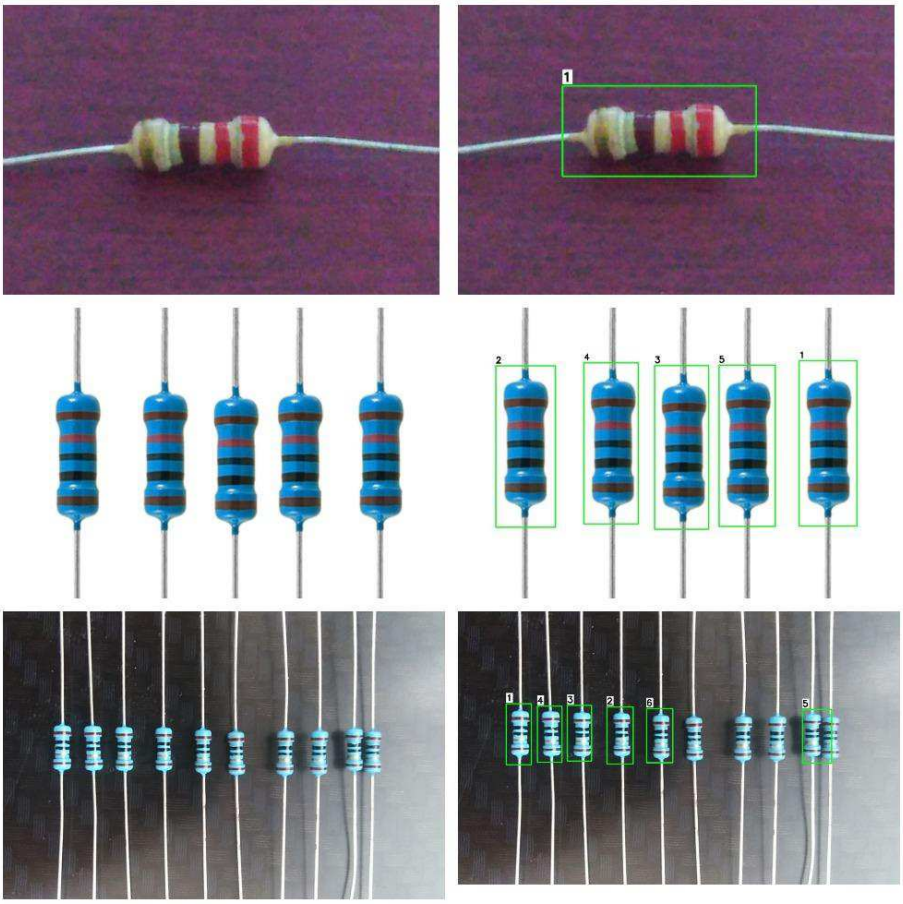

Рисунок 3.1 – Пример фотографий взятых для тестов

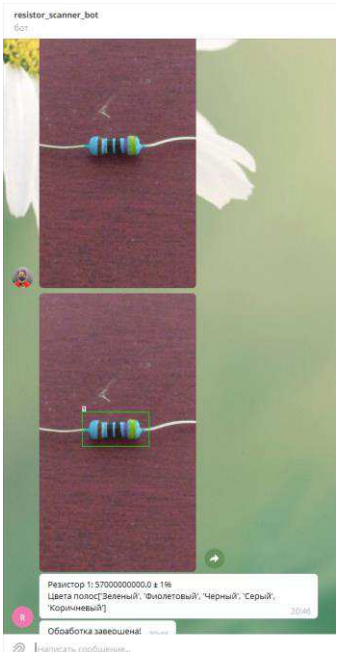

Рисунок 3.2 – Пример работы бота @resistor_scanner_bot

Проведя несколько тестов, пришли к следующим результатам:

- процент нахождения резисторов составляет 95%;
- процент верного определения цвета полосы резистора составляет 70%;
- лучше всего определяется резистор с ярко контрастирующим фоном;
- вспышка создает блики, которые препятствует распознаванию резистора;

3.4 Сравнение с другими решениями

Сравним разработанную систему с её аналогами. Для тестов возьмем тот же набор фотографий, который использовался в разделе 3.1.1. Сравнивать будем в семи критериях и результаты запишем в таблицу 3.1.

Таблица 3.1 - Сравнение результатов

Критерий	Resistor Scanner от разработчика MhTechDev	Resistor Scaner(beta) от разработчика Иoldy Labs	Предлагаемое решение
Горизонтальное распознавание	+	+	+
Вертикальное распознавание	-	-	+
Диагональное распознавание	-	-	-
Мульти распознавание	+	-	+
Процент распознавания резисторов	35%	15%	95%
Процент корректного определения цвета	40%	20%	70%
Возможность ручной отладки	+	+	-

Из таблицы 3.1 можно сделать вывод, что предлагаемое решение превосходит два других решения по критерию «Вертикальное распознавание», «Процент распознавания резисторов» и «Процент корректного определения цвета». Второй продукт не имеет возможности мульти распознавания. Предлагаемое решение проигрывает в возможности ручной отладки.

3.5 Сфера применения разработанной системы

Разработанная система является open-source проектом, и исходный код можно найти на веб-сервисе для хостинга проектов Github по ссылке https://github.com/KotikDashi/resistor_scaner.

Предлагаемое решение можно использовать в различных ситуациях:

1) Проверка качества печатных плат. При изготовлении печатных плат необходимо осуществлять проверку качества всех элементов и созданное решение позволяет решать одну из задач проверки.

2) Диагностика и ремонт печатных плат. В случае неполадок с печатной платой, требуется произвести полную диагностику печатной платы. Данное решение позволяет облегчить такую трудоемкую работу.

3) Индивидуальное использование в собственных проектах. Также данным решением может воспользоваться каждый человек, который занимается постройкой своих проектов с использованием резисторов с цветовой маркировкой.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была разработана автоматическая система распознавания номинала резистора.

С помощью открытой библиотеки Tensorflow была переобучена нейронная сеть модели ssdlite_mobilenet_v2_coco с 95% точностью нахождения резистора.

Были разработаны алгоритмы для выделения образа резистора с общей фотографии с помощью библиотек PIL, OpenCV2.

Также был разработан алгоритм идентификации цвета полос резистора с фотографии с помощью цветового пространства CIE lab и формулы цветового отличия.

Для презентации проекта был разработан telegram бот с названием @resistor_scanner_bot.

Все скрипты для проекта были написаны на языке Python версии 3.8.

Проект был выложен на веб-хостинге для IT-проектов Github для дальнейшего бесплатного распространения.

Таким образом, все задачи, поставленные в ходе выпускной квалификационной работы, были решены и цель работы достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Машинное зрение. Что это и как им пользоваться? Обработка изображений оптического источника / Коллективный блог Хабрахабр [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/350918/

2 ГОСТ 11076 – 69. Конденсаторы и резисторы. Обозначения электрических параметров. Взамен ГОСТ 11076 – 64; введ. 01.01.1970. – Москва: Издательство стандартов, 1978. – 5 с.

3 ГОСТ 8.417 – 2002. Единицы величин. Взамен ГОСТ 8.417-81; введ 01.09.2003. - Москва: Госстандарт России, 2003. – 24 с.

4 Пестриков, В. М. / Энциклопедия радиолюбителя / Пестриков В. М. – Москва: Наука и техника, 2006. – 432 с.

5 ГОСТ 28883 – 90. Коды для маркировки резисторов и конденсаторов. Введ. 01.01.1992. – Москва: Издательство стандартов, 1992. – 20 с.

6 Нейронные сети // Большая российская энциклопедия [Электронный ресурс]. – Режим доступа: https://bigenc.ru/technology_and_technique/text/4114009

7 Николенко, С. Глубокое обучение: Уч. пособие / С. Николенко, А. Кадурин, Е. Архангельская - Санкт-петербург: Питер, 2020, 480 с.

8 TensorFlow Lite API [Электронный ресурс]. – Режим доступа: https://www.tensorflow.org/lite/api_docs?hl=en

9 Google coollaboratory [Электронный ресурс]. – Режим доступа: https://colab.research.google.com/notebooks/intro.ipynb

10 Цветовое пространство RGB / Свободная библиотека Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/RGB

11 О цветовых пространствах / Коллективный блог Хабрахабр [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/181580/

12 ГОСТ Р 52489-2005. КОЛОРИМЕТРИЯ. Часть 1. Основные положения. Введ. 30.12.2005. – Москва: Стандартинформ, 2006. – 12 с.

13 ГОСТ Р ИСО 12647 – 1. Контроль процесса изготовления цифровых файлов, растровых цветоделений, пробных и тиражных оттисков. Введ. 02.10.2016. – Москва: Стандартинформ, 2016. – 23 с.

14 Ковалева, И. Л. Получение и обработка изображений: учебник / И. Л. Ковалева. – Минск: БИТУ, 2008. – 32 с.

15 Баннова, М. А. Методы определения цветового отличия: статья в сборнике трудов конференции / М. А. Баннова, А. Ю. Платов. – Нижний Новгород: НГАСУ, 2020. – 3 с.

16 ГОСТ Р 52490-2005. Колориметрия. Часть 3. Расчет цветовых различий. Введ. 01.01.2007. – Москва: Стандартинформ, 2006. – 8 с.

17 Репозиторий библиотеки Color Thief [Электронный ресурс]. – Режим доступа: https://github.com/fengsp/color-thief-py

18 СТО 4.2 07 2014. Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности. – Взамен СТО 4.2 07 2012; дата введ. 09.01.2014. – Красноярск, 2014. – 60с.

ПРИЛОЖЕНИЕ А

Структура нейронной сети

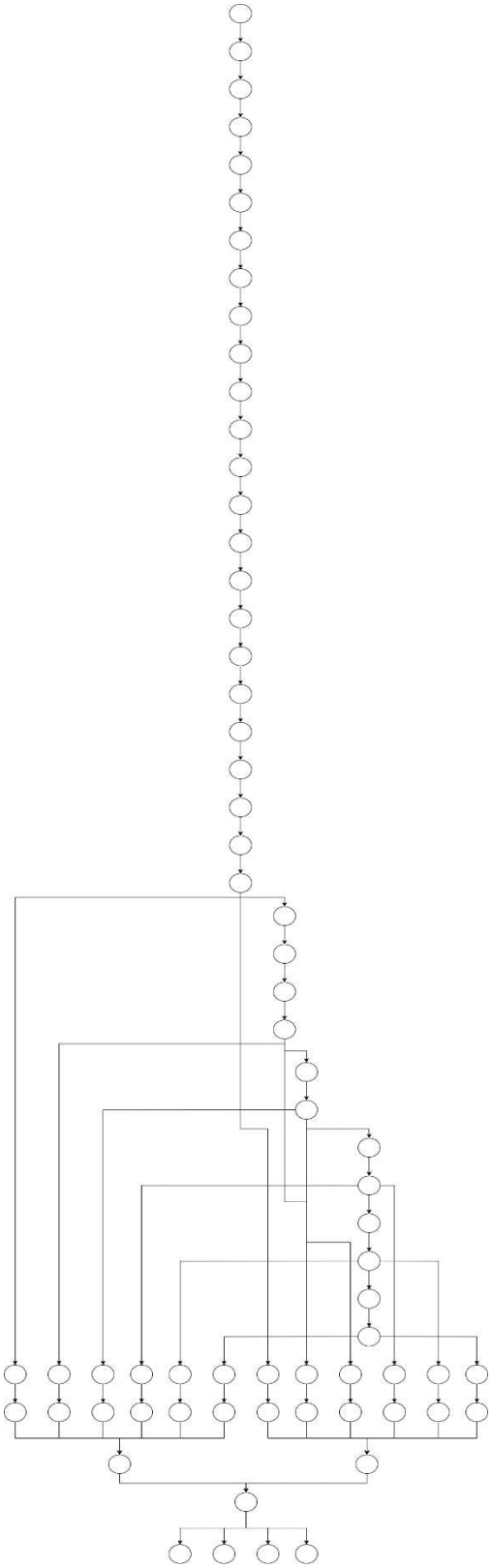

Рисунок А.1 – Структура нейронной сети

ПРИЛОЖЕНИЕ Б

Скрипты предлагаемого решения

```
# main.py

# импортирование библиотек
import os
import telebot
import shutil
from crop_script import crop1, crop2, crop3
from color_script import Search
from crop_script import edges

TOKEN = '1864374357:AAGyJfkHyIuPKoUqIpHZpHnSG6eyacrI7uk'  # токен бота
bot = telebot.TeleBot(TOKEN)



def remove_folder_contents(path):  #Функция удаления данных из папки
    shutil.rmtree(path)
    os.makedirs(path)



@bot.message_handler(content_types=['photo'])
def handle_docs_photo(message):
    # удаление всех файлов с прошлого фото
    remove_folder_contents('photos')
    remove_folder_contents('crop_photos/crop_photos1')
    remove_folder_contents('crop_photos/crop_photos2')
    remove_folder_contents('crop_photos/crop_photos3')
    remove_folder_contents('crop_photos/edges')
    remove_folder_contents('crop_photos/convex')
    os.system(r'nul>result.txt')

    # начало алгоритма
    try:
        file_info = bot.get_file(message.photo[len(message.photo) - 1].file_id)
        downloaded_file = bot.download_file(file_info.file_path)
        src = file_info.file_path
        with open(src, 'wb') as new_file:
            new_file.write(downloaded_file)
```

```
                print(src)
            #  проход фото через нейронку
            command = "python TFLite_detection_image.py --modeldir=TFLite_model --
image=" + src
            os.system(command)

            # обрезка фото
            photo = open('end_image.jpg', 'rb')
            crop1.Crop(src)
            f = open('count.txt')
            count = f.read()
            edges.save_edges(count)
            crop2.Crop(count)
            # crop3.Crop(count)

            # отправка фото
            bot.send_photo(message.chat.id, photo)

            # поиск полос и вычисление результата
            Search.search_strip(count)

            f = open('result.txt')
            text = f.read()

            # отправка результата в чат
            bot.send_message(message.from_user.id, text)
            bot.send_message(message.from_user.id, "Обработка завершена!")

        except Exception as e:
            # печать ошибки
            bot.reply_to(message, e)


bot.polling()

#TFLite_detection_image.py
import argparse
import glob
import importlib.util
import os
```

```
import sys

import cv2
import numpy as np

# Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
                    required=True)
parser.add_argument('--graph', help='Name of the .tflite file, if different than
detect.tflite',
                    default='detect.tflite')
parser.add_argument('--labels', help='Name of the labelmap file, if different than
labelmap.txt',
                    default='labelmap.txt')
parser.add_argument('--threshold',   help='Minimum   confidence   threshold   for
displaying detected objects',
                    default=0.45)
parser.add_argument('--image',
                    help='Name of the single image to perform detection on. To run
detection on multiple images, use --imagedir',
                    default=None)
parser.add_argument('--imagedir',
                    help='Name of the folder containing images to perform detection
on. Folder must contain only images.',
                    default=None)
parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to speed up
detection',
                    action='store_true')

args = parser.parse_args()

MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
min_conf_threshold = float(args.threshold)
use_TPU = args.edgetpu

# Parse input image name and directory.
IM_NAME = args.image
```

```
IM_DIR = args.imagedir

# If both an image AND a folder are specified, throw an error
if (IM_NAME and IM_DIR):
    print(
        'Error! Please only use the --image argument or the --imagedir argument, not both. Issue "python TFLite_detection_image.py -h" for help.')
    sys.exit()

# If neither an image or a folder are specified, default to using 'test1.jpg' for image name
if (not IM_NAME and not IM_DIR):
    IM_NAME = 'test1.jpg'

# Import TensorFlow libraries
# If tflite_runtime is installed, import interpreter from tflite_runtime, else import from regular tensorflow
# If using Coral Edge TPU, import the load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter

    if use_TPU:
        from tflite_runtime.interpreter import load_delegate
else:
    from tensorflow.lite.python.interpreter import Interpreter

    if use_TPU:
        from tensorflow.lite.python.interpreter import load_delegate

# If using Edge TPU, assign filename for Edge TPU model
if use_TPU:
    # If user has specified the name of the .tflite file, use that name, otherwise use default 'edgetpu.tflite'
    if (GRAPH_NAME == 'detect.tflite'):
        GRAPH_NAME = 'edgetpu.tflite'

# Get path to current working directory
CWD_PATH = os.getcwd()
```

```
        # Define path to images and grab all image filenames
        if IM_DIR:
            PATH_TO_IMAGES = os.path.join(CWD_PATH, IM_DIR)
            images = glob.glob(PATH_TO_IMAGES + '/*')

        elif IM_NAME:
            PATH_TO_IMAGES = os.path.join(CWD_PATH, IM_NAME)
            images = glob.glob(PATH_TO_IMAGES)

        # Path to .tflite file, which contains the model that is used for object detection
        PATH_TO_CKPT = os.path.join(CWD_PATH, MODEL_NAME, GRAPH_NAME)

        # Path to label map file
        PATH_TO_LABELS = os.path.join(CWD_PATH, MODEL_NAME, LABELMAP_NAME)

        # Load the label map
        with open(PATH_TO_LABELS, 'r') as f:
            labels = [line.strip() for line in f.readlines()]

        # First label is '???', which has to be removed.
        if labels[0] == '???':
            del (labels[0])

        # Load the Tensorflow Lite model.
        # If using Edge TPU, use special load_delegate argument
        if use_TPU:
            interpreter = Interpreter(model_path=PATH_TO_CKPT,

experimental_delegates=[load_delegate('libedgetpu.so.1.0')])
            print(PATH_TO_CKPT)
        else:
            interpreter = Interpreter(model_path=PATH_TO_CKPT)

        interpreter.allocate_tensors()

        # Get model details
        input_details = interpreter.get_input_details()
        output_details = interpreter.get_output_details()
        height = input_details[0]['shape'][1]
        width = input_details[0]['shape'][2]
```

```
        floating_model = (input_details[0]['dtype'] == np.float32)

        input_mean = 127.5
        input_std = 127.5

        # Loop over every image and perform detection
        for image_path in images:

            # Load image and resize to expected shape [1xHxWx3]
            image = cv2.imread(image_path)
            image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            imH, imW, _ = image.shape
            image_resized = cv2.resize(image_rgb, (width, height))
            input_data = np.expand_dims(image_resized, axis=0)

            # Normalize pixel values if using a floating model (i.e. if model is non-
quantized)
            if floating_model:
                input_data = (np.float32(input_data) - input_mean) / input_std

            # Perform the actual detection by running the model with the image as input
            interpreter.set_tensor(input_details[0]['index'], input_data)
            interpreter.invoke()

            # Retrieve detection results
            boxes = interpreter.get_tensor(output_details[0]['index'])[0]  # Bounding box
coordinates of detected objects
            classes = interpreter.get_tensor(output_details[1]['index'])[0]  # Class index
of detected objects
            scores = interpreter.get_tensor(output_details[2]['index'])[0]  # Confidence
of detected objects
            # num = interpreter.get_tensor(output_details[3]['index'])[0]  # Total number
of detected objects (inaccurate and not needed)

            # Loop over all detections and draw detection box if confidence is above minimum
threshold
            count = 1
            f = open('count.txt', 'w')
            pos = open('pos.txt', 'w')
```

```
            for i in range(len(scores)):
                if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):
                    # Get bounding box coordinates and draw box
                    # Interpreter can return coordinates that are outside of image
dimensions, need to force them to be within image using max() and min()
                    ymin = int(max(1, (boxes[i][0] * imH)))
                    xmin = int(max(1, (boxes[i][1] * imW)))
                    ymax = int(min(imH, (boxes[i][2] * imH)))
                    xmax = int(min(imW, (boxes[i][3] * imW)))

                    ymin = int(ymin - (ymax - ymin) * 0.05)
                    ymax = int(ymax + (ymax - ymin) * 0.05)
                    xmin = int(xmin - (xmax - xmin) * 0.05)
                    xmax = int(xmax + (xmax - xmin) * 0.05)

                    if (xmin > 0) and (xmax < imW) and (ymin > 0) and (ymax < imH) and
((ymax - ymin) > imH * 0.02) and (
                            (xmax - xmin) > imW * 0.02):
                        devx = int((xmax - xmin) * 0.05)
                        devy = int((ymax - ymin) * 0.05)
                        pos.write(str(ymin) + '\n')
                        pos.write(str(xmin) + '\n')
                        pos.write(str(ymax) + '\n')
                        pos.write(str(xmax) + '\n')
                        cv2.rectangle(image, (xmin, ymin), (xmax, ymax), (10, 255, 0), 2)
                        label = str(count)
                        labelSize,        baseLine        =        cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2)
                        label_ymin = max(ymin, labelSize[1] + 10)
                        cv2.rectangle(image, (xmin, label_ymin - labelSize[1] - 10),
                                      (xmin + labelSize[0], label_ymin + baseLine - 10),
(255, 255, 255),
                                      cv2.FILLED)
                        cv2.putText(image,    label,    (xmin,    label_ymin    -    7),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0),
                                    2)
                        count += 1

            count -= 1
            f.write(str(count) + '\n')
```

```
    # All the results have been drawn on the image, now display the image
    cv2.imwrite('end_image.jpg', image)

    # Press any key to continue to next image, or press 'q' to quit
    if cv2.waitKey(0) == ord('q'):
        break

# Clean up
cv2.destroyAllWindows()

#CalculatingTheValue.py
# значащие цифры
def significant_numbers(value):
    if value == "Черный":
        return '0'
    if value == "Коричневый":
        return '1'
    if value == "Красный":
        return '2'
    if value == "Оранжевый":
        return '3'
    if value == "Желтый":
        return '4'
    if value == "Зеленый":
        return '5'
    if value == "Голубой":
        return '6'
    if value == "Фиолетовый":
        return '7'
    if value == "Серый":
        return '8'
    if value == "Белый":
        return '9'


# множитель
def multiplier(value):
    if value == "Черный":
        return '1'
```

```
    if value == "Коричневый":
        return '10'
    if value == "Красный":
        return '100'
    if value == "Оранжевый":
        return '1000'
    if value == "Желтый":
        return '10000'
    if value == "Зеленый":
        return '100000'
    if value == "Голубой":
        return '1000000'
    if value == "Фиолетовый":
        return '10000000'
    if value == "Серый":
        return '100000000'
    if value == "Белый":
        return '1000000000'
    if value == "Серебряный":
        return '0.01'
    if value == "Золотой":
        return '0.1'


# допуск
def Engineering_tolerance(value):
    if value == "Черный":
        return '± 20%'
    if value == "Коричневый":
        return '± 1%'
    if value == "Красный":
        return '± 2%'
    if value == "Зеленый":
        return '± 0.5%'
    if value == "Голубой":
        return '± 0.25%'
    if value == "Фиолетовый":
        return '± 0.1%'
    if value == "Серый":
        return '± 0.05%'
```

```
    if value == "Серебряный":
        return '± 10%'
    if value == "Золотой":
        return '± 5%'


# ТК
def ТК(value):
    if value == "Коричневый":
        return '100'
    if value == "Красный":
        return '50'
    if value == "Оранжевый":
        return '15'
    if value == "Желтый":
        return '25'
    if value == "Голубой":
        return '10'
    if value == "Фиолетовый":
        return '5'
    if value == "Белый":
        return '1'


# поиск значения
def resistor_value(arr, k):
    f = open('result.txt', 'a')
    try:
        len_arr = len(arr)
        value = ''
        if len_arr == 3:
            for i in range(2):
                value += significant_numbers(arr[i])
            value = int(value) * float(multiplier(arr[2]))

        if len_arr == 4:
            for i in range(2):
                value = value + significant_numbers(arr[i])
            value = int(value) * float(multiplier(arr[2]))
            value = str(round(value, 4)) + ' ' + Engineering_tolerance(arr[3])
```

```
                print(value)

            if len_arr == 5:
                for i in range(3):
                    value += significant_numbers(arr[i])
                value = int(value) * float(multiplier(arr[3]))
                value = str(round(value, 4)) + ' ' + Engineering_tolerance(arr[4])

            if len_arr == 6:
                for i in range(3):
                    value += significant_numbers(arr[i])
                value = int(value) * float(multiplier(arr[3]))
                value = str(round(value, 4)) + ' ' + Engineering_tolerance(arr[4])

            if len_arr != 6:
                f.write('Резистор {0}: '.format(k) + str(value) + '\n')
            else:
                f.write('Резистор {0}: '.format(k) + str(value) + ', ТКС = ' +
str(TK(arr[5])) + '\n')

                f.write('Цвета полос' + str(arr) + '\n\n')
        except:
            f.write('Резистор {0}: ошибка\n'.format(k))
            f.write('Цвета полос' + str(arr) + '\n\n')
        f.close()

    #color_search.py

    from color_script import rgb_lab

    # color_rgb

    silver = [[192, 192, 192]]
    gold = [[255, 215, 0], ]
    black = [[0, 0, 0], [5, 0, 4], [34, 48, 48]]
    brown = [[150, 75, 0], [120, 67, 58], [109, 59, 62], [52, 7, 5], [96, 45, 24]]
    red = [[112, 57, 54], [137, 29, 44]]
    orange = [[255, 165, 0], [243, 73, 21]]
    yellow = [[255, 255, 0]]
    green = [[0, 128, 0], [49, 89, 56], [123, 149, 101]]
```

```
purple = [[135, 0, 255]]
gray = [[128, 128, 128], [120, 118, 124]]
white = [[255, 255, 255], [226, 214, 210]]
blue = [[29, 42, 89], [25, 61, 107], [0, 0, 255]]

# color_lab

silver_lab = []
gold_lab = []
black_lab = []
brown_lab = []
red_lab = []
orange_lab = []
yellow_lab = []
green_lab = []
purple_lab = []
gray_lab = []
white_lab = []
blue_lab = []

for i in silver:
    silver_lab.append(rgb_lab.rgb2lab(i))

for i in gold:
    gold_lab.append(rgb_lab.rgb2lab(i))

for i in black:
    black_lab.append(rgb_lab.rgb2lab(i))

for i in brown:
    brown_lab.append(rgb_lab.rgb2lab(i))

for i in red:
    red_lab.append(rgb_lab.rgb2lab(i))

for i in orange:
    orange_lab.append(rgb_lab.rgb2lab(i))

for i in yellow:
    yellow_lab.append(rgb_lab.rgb2lab(i))
```

```
        for i in green:
            green_lab.append(rgb_lab.rgb2lab(i))

        for i in purple:
            purple_lab.append(rgb_lab.rgb2lab(i))

        for i in gray:
            gray_lab.append(rgb_lab.rgb2lab(i))

        for i in white:
            white_lab.append(rgb_lab.rgb2lab(i))

        for i in blue:
            blue_lab.append(rgb_lab.rgb2lab(i))

        my_dict = {"Серебряный ": silver_lab, "Золотой": gold_lab, "Черный": black_lab,
"Коричневый": brown_lab,
                   "Красный": red_lab,
                   "Оранжевый": orange_lab, "Желтый": yellow_lab,
                   "Зеленый": green_lab, "Фиолетовый": purple_lab, "Серый": gray_lab,
"Белый": white_lab, "Синий": blue_lab}


        # поиск цвета
        def color_search(pixel):
            lab = rgb_lab.rgb2lab(pixel)
            dE_Arr_out = []
            keys_arr = []
            for i in my_dict:
                dE_Arr = []
                for j in my_dict[i]:
                    dE_Arr.append(rgb_lab.CIEDE2000(lab, j))
                dE_Arr_out.append(dE_Arr)
                keys_arr.append(i)

            min_dE = 101
            min_index = -1
            for i in range(len(dE_Arr_out)):
                if min(dE_Arr_out[i]) < min_dE:
```

```
            min_dE = min(dE_Arr_out[i])
            min_index = i

    return min_dE, keys_arr[min_index]


# поиск цвета с цветом корпуса
def color_search2(pixel, color_case):
    my_dict["цвет корпуса"] = [rgb_lab.rgb2lab(color_case)]
    lab = rgb_lab.rgb2lab(pixel)
    dE_Arr_out = []
    keys_arr = []
    for i in my_dict:
        dE_Arr = []
        for j in my_dict[i]:
            dE_Arr.append(rgb_lab.CIEDE2000(lab, j))
        dE_Arr_out.append(dE_Arr)
        keys_arr.append(i)

    min_dE = 101
    min_index = -1
    for i in range(len(dE_Arr_out)):
        if min(dE_Arr_out[i]) < min_dE:
            min_dE = min(dE_Arr_out[i])
            min_index = i

    return min_dE, keys_arr[min_index]

# dominant_color.py
from colorthief import ColorThief


# поиск доминантного цвета
def color_thief(src):
    color_thief = ColorThief(src)
    # get the dominant color
    dominant_color = color_thief.get_color(quality=1)
    # build a color palette
    return dominant_color
```

```
#rgb_lab.py
import math


# Calculates CIEDE2000 color distance between two CIE L*a*b* colors
def CIEDE2000(Lab_1, Lab_2):
    C_25_7 = 6103515625  # 25**7

    L1, a1, b1 = Lab_1[0], Lab_1[1], Lab_1[2]
    L2, a2, b2 = Lab_2[0], Lab_2[1], Lab_2[2]
    C1 = math.sqrt(a1 ** 2 + b1 ** 2)
    C2 = math.sqrt(a2 ** 2 + b2 ** 2)
    C_ave = (C1 + C2) / 2
    G = 0.5 * (1 - math.sqrt(C_ave ** 7 / (C_ave ** 7 + C_25_7)))

    L1_, L2_ = L1, L2
    a1_, a2_ = (1 + G) * a1, (1 + G) * a2
    b1_, b2_ = b1, b2

    C1_ = math.sqrt(a1_ ** 2 + b1_ ** 2)
    C2_ = math.sqrt(a2_ ** 2 + b2_ ** 2)

    if b1_ == 0 and a1_ == 0:
        h1_ = 0
    elif a1_ >= 0:
        h1_ = math.atan2(b1_, a1_)
    else:
        h1_ = math.atan2(b1_, a1_) + 2 * math.pi

    if b2_ == 0 and a2_ == 0:
        h2_ = 0
    elif a2_ >= 0:
        h2_ = math.atan2(b2_, a2_)
    else:
        h2_ = math.atan2(b2_, a2_) + 2 * math.pi

    dL_ = L2_ - L1_
    dC_ = C2_ - C1_
    dh_ = h2_ - h1_
    if C1_ * C2_ == 0:
```

```
            dh_ = 0
        elif dh_ > math.pi:
            dh_ -= 2 * math.pi
        elif dh_ < -math.pi:
            dh_ += 2 * math.pi
        dH_ = 2 * math.sqrt(C1_ * C2_) * math.sin(dh_ / 2)

        L_ave = (L1_ + L2_) / 2
        C_ave = (C1_ + C2_) / 2

        _dh = abs(h1_ - h2_)
        _sh = h1_ + h2_
        C1C2 = C1_ * C2_

        if _dh <= math.pi and C1C2 != 0:
            h_ave = (h1_ + h2_) / 2
        elif _dh > math.pi and _sh < 2 * math.pi and C1C2 != 0:
            h_ave = (h1_ + h2_) / 2 + math.pi
        elif _dh > math.pi and _sh >= 2 * math.pi and C1C2 != 0:
            h_ave = (h1_ + h2_) / 2 - math.pi
        else:
            h_ave = h1_ + h2_

        T = 1 - 0.17 * math.cos(h_ave - math.pi / 6) + 0.24 * math.cos(2 * h_ave) +
0.32 * math.cos(
            3 * h_ave + math.pi / 30) - 0.2 * math.cos(4 * h_ave - 63 * math.pi / 180)

        h_ave_deg = h_ave * 180 / math.pi
        if h_ave_deg < 0:
            h_ave_deg += 360
        elif h_ave_deg > 360:
            h_ave_deg -= 360
        dTheta = 30 * math.exp(-(((h_ave_deg - 275) / 25) ** 2))

        R_C = 2 * math.sqrt(C_ave ** 7 / (C_ave ** 7 + C_25_7))
        S_C = 1 + 0.045 * C_ave
        S_H = 1 + 0.015 * C_ave * T

        Lm50s = (L_ave - 50) ** 2
        S_L = 1 + 0.015 * Lm50s / math.sqrt(20 + Lm50s)
```

```
    R_T = -math.sin(dTheta * math.pi / 90) * R_C

    k_L, k_C, k_H = 1, 1, 1

    f_L = dL_ / k_L / S_L
    f_C = dC_ / k_C / S_C
    f_H = dH_ / k_H / S_H

    dE_00 = math.sqrt(f_L ** 2 + f_C ** 2 + f_H ** 2 + R_T * f_C * f_H)
    return dE_00


def delta_E(lab1, lab2):
    l1 = lab1[0]
    a1 = lab1[1]
    b1 = lab1[2]

    l2 = lab2[0]
    a2 = lab2[1]
    b2 = lab2[2]

    dE = math.sqrt(pow(l2 - l1, 2) + pow(a2 - a1, 2) + pow(b2 - b1, 2))
    return dE


# конвертация rgb в lab
def rgb2lab(inputColor):
    num = 0
    RGB = [0, 0, 0]

    for value in inputColor:
        value = float(value) / 255

        if value > 0.04045:
            value = ((value + 0.055) / 1.055) ** 2.4
        else:
            value = value / 12.92

        RGB[num] = value * 100
        num = num + 1
```

```
    XYZ = [0, 0, 0, ]

    X = RGB[0] * 0.4124 + RGB[1] * 0.3576 + RGB[2] * 0.1805
    Y = RGB[0] * 0.2126 + RGB[1] * 0.7152 + RGB[2] * 0.0722
    Z = RGB[0] * 0.0193 + RGB[1] * 0.1192 + RGB[2] * 0.9505
    XYZ[0] = round(X, 4)
    XYZ[1] = round(Y, 4)
    XYZ[2] = round(Z, 4)

    # Observer= 2°, Illuminant= D65
    XYZ[0] = float(XYZ[0]) / 95.047  # ref_X =  95.047
    XYZ[1] = float(XYZ[1]) / 100.0  # ref_Y = 100.000
    XYZ[2] = float(XYZ[2]) / 108.883  # ref_Z = 108.883

    num = 0
    for value in XYZ:

        if value > 0.008856:
            value = value ** (0.3333333333333333)
        else:
            value = (7.787 * value) + (16 / 116)

        XYZ[num] = value
        num = num + 1

    Lab = [0, 0, 0]

    L = (116 * XYZ[1]) - 16
    a = 500 * (XYZ[0] - XYZ[1])
    b = 200 * (XYZ[1] - XYZ[2])

    Lab[0] = round(L, 4)
    Lab[1] = round(a, 4)
    Lab[2] = round(b, 4)

    return Lab
#Search.py
from color_script import strip
from color_script import CalculatingTheValue
```

```
#поиск полос на изображении
def search_strip(count):
    for i in range(1, int(count) + 1, 1):
        src = 'crop_photos/crop_photos2/{0}.jpg'.format(i)
        arr_result = strip.search(src)
        CalculatingTheValue.resistor_value(arr_result, i)

#strip.py
from PIL import Image
from color_script import color_search, dominant_color


# поиск полос на изображении
def search(src):
    img = Image.open(src)
    w, h = img.size
    dc = [dominant_color.color_thief(src), img.getpixel((1, int(h / 2))),
img.getpixel((w - 2, int(h / 2)))]

    length = 0
    arr = []
    arr_result = []

    for x in range(w):
        xy = (x, int(h * 0.7))
        pixel = img.getpixel(xy)
        result1 = color_search.color_search2(pixel, dc[0])
        result2 = color_search.color_search2(pixel, dc[1])
        result3 = color_search.color_search2(pixel, dc[2])

        if (result1[1] == 'цвет корпуса') or (result2[1] == 'цвет корпуса') or
(result3[1] == 'цвет корпуса'):
            if length > 0.05 * w:
                arr.append(int(x - (length / 2)))
            length = 0
        else:
            length += 1

    if len(arr) > 0:
```

```
        if arr[0] > w - arr[len(arr) - 1]:
            arr = reversed(arr)

        for x in arr:
            croparea = (x - 2, 10, x + 2, h - 10)
            cropped = img.crop(croparea)
            cropped.save('temp.jpg')
            pixel = dominant_color.color_thief('temp.jpg')
            result0 = color_search.color_search(pixel)
            print(pixel, result0[1])
            arr_result.append(result0[1])
    return arr_result
#convex.py
import numpy as np
from skimage.morphology import convex_hull_image
from skimage import io
import skimage
from PIL import Image, ImageEnhance


def convex_crop(src_img, src_edge, src_crop, src_convex):
    image = np.array(Image.open(src_edge))  # открытие изображения
    chull = convex_hull_image(image)  # создание convex_hull

    io.imsave(src_convex, skimage.img_as_ubyte(chull))  # сохранение изображения

    img = Image.open(src_convex)
    w, h = img.size

    start = 0
    end = 0

    # поиск левого и правого края
    for x in range(w):
        xy = (x, int(h * 0.75))
        result1 = img.getpixel(xy)
        xy = (x, int(h * 0.25))
        result2 = img.getpixel(xy)
        if result1 > 200 and result2 > 200:
            start = x
```

```
            break

    for x in reversed(range(w)):
        xy = (x, int(h * 0.75))
        result1 = img.getpixel(xy)
        xy = (x, int(h * 0.25))
        result2 = img.getpixel(xy)
        if result1 > 200 and result2 > 200:
            end = x
            break

    [rows, columns] = np.where(chull)
    row1 = min(rows)
    row2 = max(rows)

    # обрезка изображения
    img = Image.open(src_img)
    area = (start, row1, end, row2)
    cropped_img = img.crop(area)
    enhancer = ImageEnhance.Contrast(cropped_img)
    cropped_img = enhancer.enhance(1.5)
    cropped_img.save(src_crop)

# crop1.py

import cv2


# обрезка резисторов из общей фотграфии
def Crop(src):
    image = cv2.imread(src)
    f = open('count.txt')
    count = f.read()
    f.close()
    file = open('pos.txt')

    for i in range(1, int(count) + 1, 1):
        ymin = int(file.readline()) - 3
        xmin = int(file.readline()) + 3
        ymax = int(file.readline()) + 3
```

```
                xmax = int(file.readline()) - 3
                crop_img = image[ymin:ymax, xmin:xmax]
                h, w, _ = crop_img.shape
                if h > w:
                    crop_img = cv2.rotate(crop_img, cv2.ROTATE_90_CLOCKWISE)
                cv2.imwrite('crop_photos/crop_photos1/' + str(i) + '.jpg', crop_img)
            file.close()

        # crop2.py

        from crop_script import convex

        #обрезка по маске convex
        def Crop(count):
            for i in range(1, int(count) + 1, 1):
                src_photo = 'crop_photos/crop_photos1/{0}.jpg'.format(i)
                src_edge = 'crop_photos/edges/{0}.jpg'.format(i)
                src_crop = 'crop_photos/crop_photos2/{0}.jpg'.format(i)
                src_convex = 'crop_photos/convex/{0}.jpg'.format(i)
                convex.convex_crop(src_img=src_photo,                    src_edge=src_edge,
src_crop=src_crop, src_convex=src_convex)

        # -*- coding: utf8 -*-

        # crop3.py
        from PIL import Image, ImageEnhance
        from color_script import dominant_color, color_search


        # обрезка слева и права резистора
        def Crop(count):
            for i in range(1, int(count) + 1, 1):
                src_photo = 'crop_photos/crop_photos2/{0}.jpg'.format(i)
                src_crop = 'crop_photos/crop_photos3/{0}.jpg'.format(i)
                image = Image.open(src_photo)
                dc = dominant_color.color_thief(src_photo)
                w, h = image.size
                start = 0
                end = w
```

```
        for x in range(w):
            pixel1 = image.getpixel((x, int(h * 0.25)))
            pixel2 = image.getpixel((x, int(h * 0.75)))
            result1 = color_search.color_search2(pixel1, dc)
            result2 = color_search.color_search2(pixel2, dc)
            if result1[1] == 'цвет корпуса' and result2[1] == 'цвет корпуса':
                start = x
                break

        for x in reversed(range(w)):
            pixel1 = image.getpixel((x, int(h * 0.3)))
            pixel2 = image.getpixel((x, int(h * 0.7)))
            result1 = color_search.color_search2(pixel1, dc)
            result2 = color_search.color_search2(pixel2, dc)
            if result1[1] == 'цвет корпуса' and result2[1] == 'цвет корпуса':
                end = x
                break

        im_crop = image.crop((start, 1, end, h))

        enhancer = ImageEnhance.Contrast(im_crop)
        im_crop = enhancer.enhance(1.5)

        im_crop.save(src_crop)

# for x in range(w):
#     count = 0
#     for y in range(h):
#         pixel = image.getpixel((x, y))
#         result = color_search.color_search2(pixel, dc)
#         if result[1] == 'цвет корпуса':
#             count += 1
#     arr.append(count / h)
#
# start = 0
# end = 0
# for i in range(len(arr)):
#     if arr[i] > 0.35:
#         start = i
#         break
```

```
#
# for i in reversed(range(len(arr))):
#     if arr[i] > 0.35:
#         end = i
#         break

# edges.py
import cv2


# поиск краев
def save_edges(count):
    for i in range(1, int(count) + 1, 1):
        image_crop = cv2.imread('crop_photos/crop_photos1/' + str(i) + '.jpg')
        img_edge = cv2.Canny(image_crop, 100, 155)
        cv2.imwrite('crop_photos/edges/' + str(i) + '.jpg', img_edge)
```

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра «Системы автоматики, автоматизированное управление
и проектирование»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ С.В. Ченцов

«16» 06 2021 г.

БАКАЛАВРСКАЯ РАБОТА

15.03.04 – Автоматизация технологических процессов и производств

СИСТЕМА АВТОМАТИЧЕСКОГО РАСПОЗНАВАНИЯ СПЕЦИФИКАЦИИ РЕЗИСТОРА

			доцент, д-р техн. наук
Руководитель	Агафонов	16.06. 2021 г.	Е. Д. Агафонов
Выпускник		16.06. 2021 г.	Д. Д. Санжитов
Нормоконтролер		16.06. 2021 г.	Т.А. Грудинова

Красноярск 2021