МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт кибербезопасности и цифровых технологий

Цифровая кафедра

РАБОТА ДОПУЩЕНА К ЗАЩИТЕ

Заведующий программой

_____ Ш.Г. Магомедов
«31» мая 2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по программе
«Программные средства решения прикладных задач искусственного интеллекта»

на тему: Модель распознавания вариант 249

Обучающийся _____ Ким Кирилл Сергеевич
подпись *Фамилия, имя, отчество*

Руководитель работы _____ _____ _____
подпись *ученая степень, должность* *Фамилия, имя, отчество*

Москва 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт кибербезопасности и цифровых технологий

Цифровая кафедра

СОГЛАСОВАНО
Заведующий программой

_____
(подпись)
Ш.Г. Магомедов

«27» мая 2024 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

Обучающийся **Ким Кирилл Сергеевич**

Направление программы **«Программные средства решения прикладных задач искусственного интеллекта»**

1. Тема выпускной квалификационной работы

Модель распознавания вариант 249

2. Цель и задачи выпускной квалификационной работы

Цель работы: Получение практических навыков в решении прикладных задач ИИ

Задачи работы: Разработать модель нейронной сети u-net для автоматической сегментации снимков

3. Этапы выпускной квалификационной работы

№ этапа	Содержание этапа выпускной квалификационной работы	Результат выполнения этапа ВКР	Срок выполнения
1.	**Постановка прикладной задачи**		—
2.	**Алгоритм решения поставленной задачи**		—
3.	**Реализация поставленной задачи**		—

4. Перечень разрабатываемых документов и графических материалов

Цель и задачи ВКР, характеристика предметной области, метод и алгоритм решения задачи, апробация (моделирование или программное обеспечение), выводы.

Задание принял к исполнению
Обучающийся: _____
подпись
«27» мая 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире с развитием технологий компьютерного зрения и машинного обучения возникает все больше возможностей для автоматизации процессов распознавания объектов на изображениях и видео. Одной из актуальных задач в этой области является распознавание бытовых вещей на снимках и видео.

Распознавание бытовых вещей имеет широкий спектр применений, начиная от улучшения процесса поиска и классификации объектов на изображениях до создания инновационных систем умного дома и робототехники. При этом, точность и скорость распознавания играют ключевую роль в эффективности таких систем.

Цель данного исследования заключается в разработке алгоритма машинного обучения, способного точно распознавать бытовые вещи на снимках и видео. Для достижения этой цели будут использованы современные методы обработки изображений, а также алгоритмы распознавания объектов.

Исследование по распознаванию бытовых вещей на снимках и видео имеет практическую значимость для различных областей, включая розничную торговлю, безопасность, медицину и другие сферы человеческой деятельности. Результаты данного исследования могут быть использованы для создания инновационных технологических решений, способствующих повышению качества жизни и улучшению рабочих процессов.

Таким образом, разработка алгоритма распознавания бытовых вещей на снимках и видео представляет собой актуальную задачу, которая имеет потенциал для широкого применения в различных областях и секторах экономики.

Задачи исследования:

1. Провести обзор существующих методов распознавания объектов на изображениях и видео.

2. Собрать и подготовить набор данных, содержащий разнообразные изображения бытовых вещей.

3. Проанализировать особенности бытовых вещей, которые могут повлиять на процесс распознавания.

4. Использовать современные методы обработки изображений для предобработки данных.

5. Обучить модель машинного обучения на подготовленном наборе данных для распознавания бытовых вещей.

6. Оценить точность и скорость работы разработанного алгоритма на тестовом наборе данных.

7. Провести сравнительный анализ с другими существующими методами распознавания объектов на изображениях и видео.

8. Предложить возможные пути улучшения алгоритма и его применение в различных областях.

1 Постановка прикладной задачи

Для процесса разметки бытовых вещей требуется скачать снимки и предоставленное программное обеспечение Supervisely.

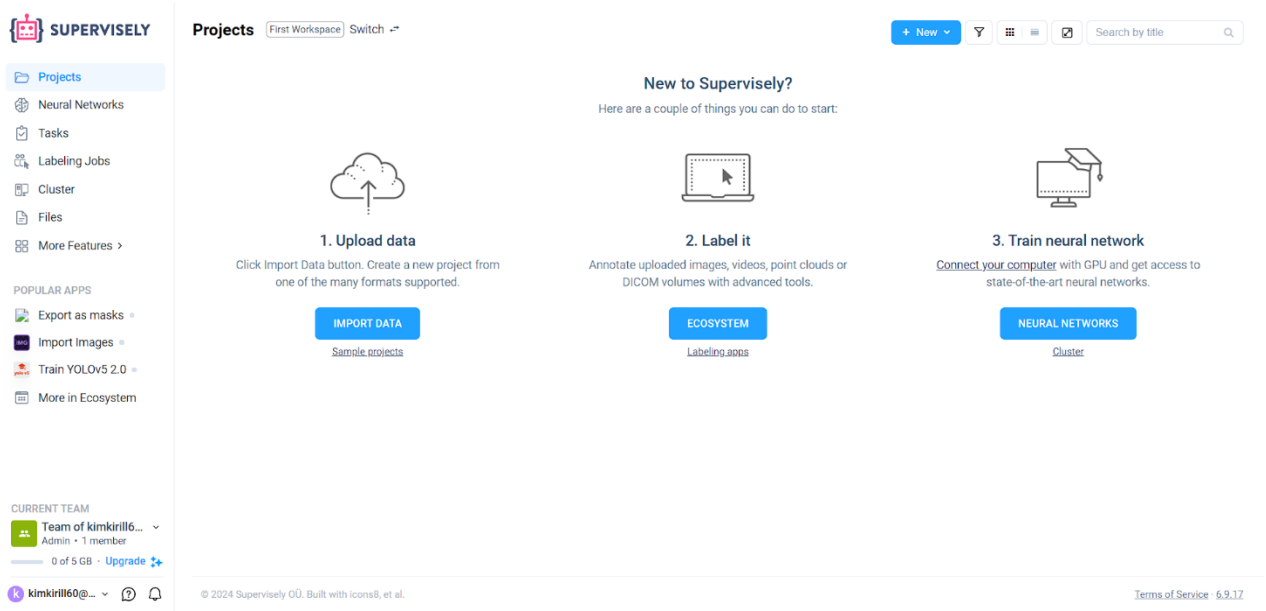

Рисунок 1.1 – Интерфейс Supervisely

Подготовить снимки и программное обеспечение. Нужно загрузить программное обеспечение, которое позволит сделать разметку снимков. После необходимо выбрать снимки вещей, которые будут использоваться для обучения модели. Далее нужно создать несколько атрибутов с разметкой radio, которые будут отвечать за отдельные категории вещей: кружка, коробка, туалетная бумага и так далее.

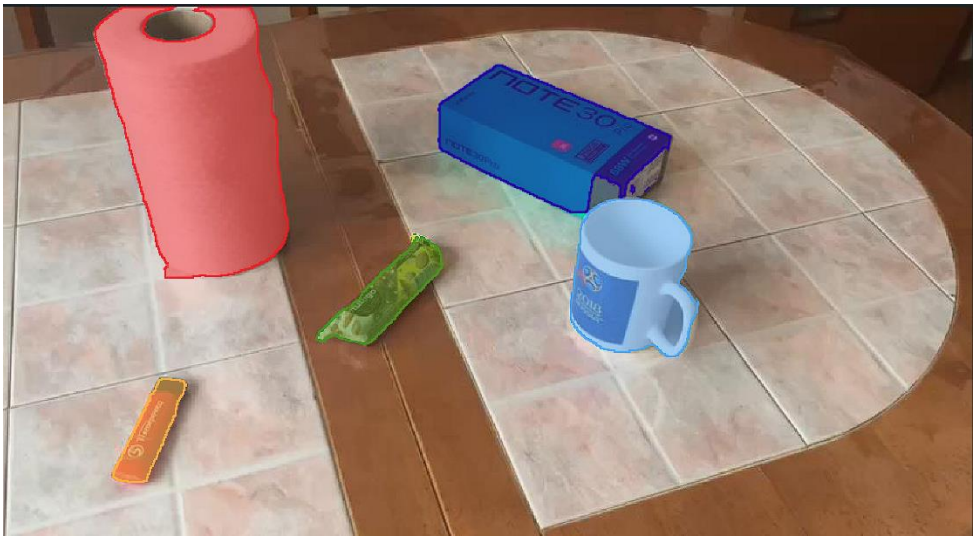

Рисунок 1.2 – Атрибуты вещей

Разметка каждого снимка - для каждого снимка необходимо отметить на изображении. Разметка производилась с помощью инструмента "Mask Pen Tool"

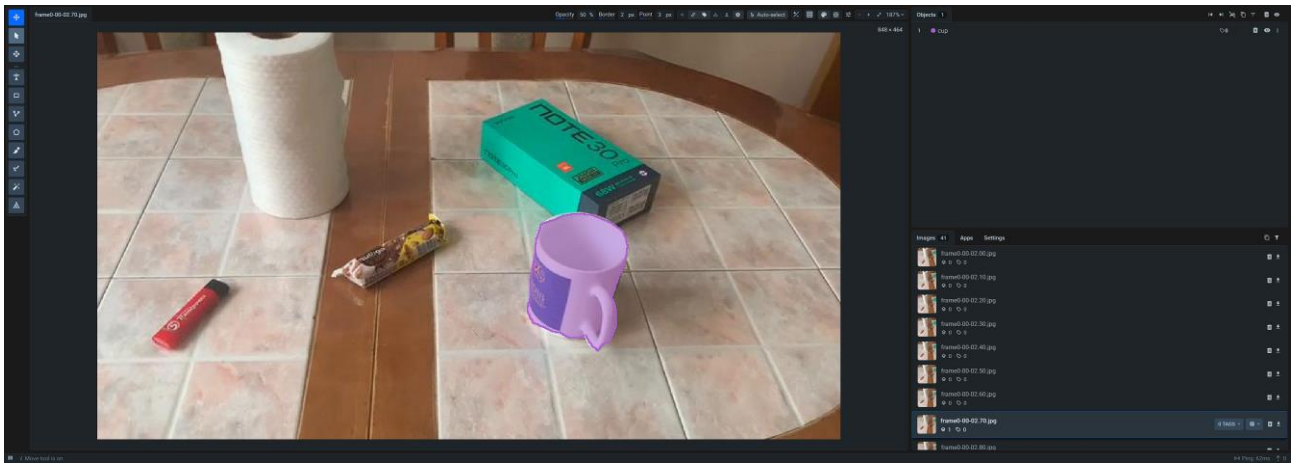

Рисунок 1.2 – Атрибут кружки

Это позволит модели научиться распознавать и классифицировать бытовые вещи на снимках.

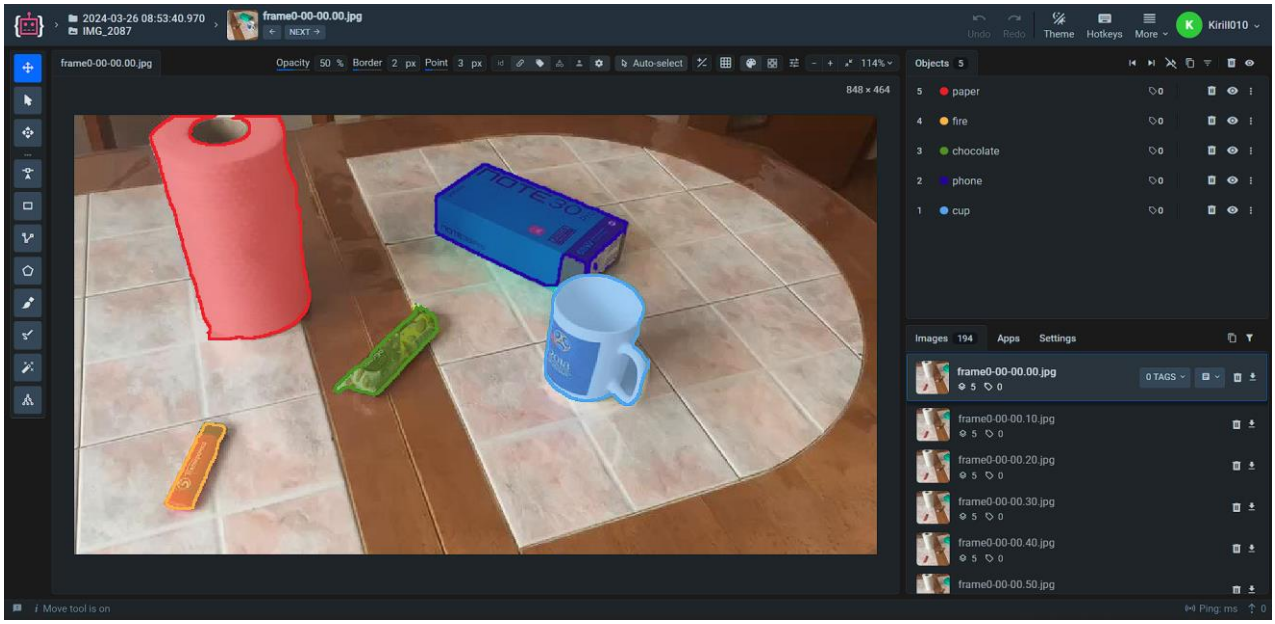

Рисунок 1.2 – Работа по разметке в программе Supervisely

Данный набор данных представляет собой фотографии бытовых вещей. На снимках могут быть обнаружены различные аномалии:

- Квадратики на столе;

2 АЛГОРИТМ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

Алгоритм решения для нейронной сети включает в себя несколько важных пунктов:

1. Определение задачи и постановка целей. В первую очередь необходимо определить задачу, решение которой требуется автоматизировать, и поставить цели, которые должны быть достигнуты при использовании программного средства искусственного интеллекта (ИИ).

2. Определение метода решения. На основе определенной задачи и целей, необходимо определить метод решения. Это может быть как метод классического машинного обучения, так и глубокого обучения.

3. Сбор и подготовка данных. Перед обучением модели необходимо собрать данные, которые будут использоваться для обучения. Затем следует предобработка данных, включающая в себя очистку исходных данных, отбор признаков, нормализацию данных и т.д.

4. Обучение модели. После предобработки данных следует обучение модели. На этом этапе выбранная модель обучается на подготовленных данных. Обучающий алгоритм позволяет модели получить опыт и на основе этого научиться предсказывать решения для данной задачи.

5. Тестирование и оценка результатов. После обучения модель должна быть протестирована на новых данных, которые ранее не использовались для обучения. Это поможет оценить качество полученной модели и ее способность к предсказанию решений в реальном времени. Оценка результата включает в себя различные метрики, такие как точность, полнота, F-мера и т.д.

6. Развитие и поддержка ИИ. Выпущенная модель требует постоянной поддержки и развития. Для этого необходимо собирать данные и периодически обучать модель заново, чтобы она могла опираться на

последние данные и сохранять высокую точность предсказаний. Кроме обучения модели, необходимо дорабатывать ее алгоритмы и функционал для повышения эффективности работы.

2.1 Структура сверточной нейронной

В сверточной нейронной сети выходы промежуточных слоев образуют матрицу или набор матриц. Так, например, на вход сверточной нейронной сети можно подавать три слоя изображения. Основными видами слоев в сверточной нейронной сети являются сверточные слои, пулинговые слои и полносвязные слои. Рассмотрим данные слои ниже:

Сверточный слой нейронной сети представляет из себя применение операции свертки к выходам с предыдущего слоя, где веса ядра свертки являются обучаемыми параметрами. Еще один обучаемый вес используется в качестве константного сдвига (англ. bias).

Пулинговый слой призван снижать размерность изображения. Исходное изображение делится на блоки размером w×h и для каждого блока вычисляется некоторая функция. Чаще всего используется функция максимума (англ. max pooling) или (взвешенного) среднего (англ. (weighted) average pooling). Обучаемых параметров у этого слоя нет. Основные цели пулингового слоя:

- уменьшение изображения, чтобы последующие свертки оперировали над большей областью исходного изображения;
- увеличение инвариантности выхода сети по отношению к малому переносу входа;
- ускорение вычислений.

Inception module — это специальный слой нейронной сети, который был предложен в работе, в которой была представлена сеть GoogLeNet. Основная цель этого модуля заключается в следующем. Авторы предположили, что каждый элемент предыдущего слоя соответствует

определенной области исходного изображения. Каждая свертка по таким элементам будет увеличивать область исходного изображения, пока элементы на последних слоях не будут соответствовать всему изображению целиком. Однако, если с какого-то момента все свертки станут размером 1×1, то не найдется элементов, которые покрывали бы все исходное изображение, поэтому было бы невозможно находить большие признаки на рисунке 2.1. Чтобы решить эту проблему, авторы предложили так называемый inception module — конкатенацию выходов для сверток размера 1×1, 3×3, 5×5, а также операции max pooling'a с ядром 3×3.

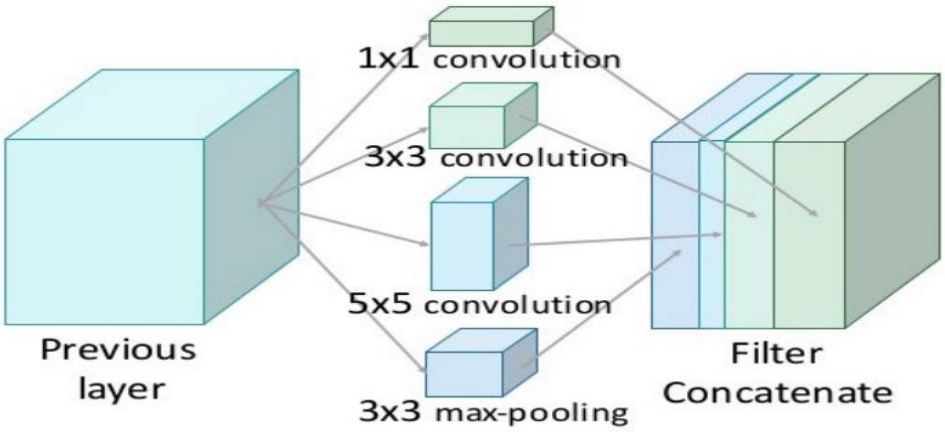

Рисунок 2.1 — Inception module

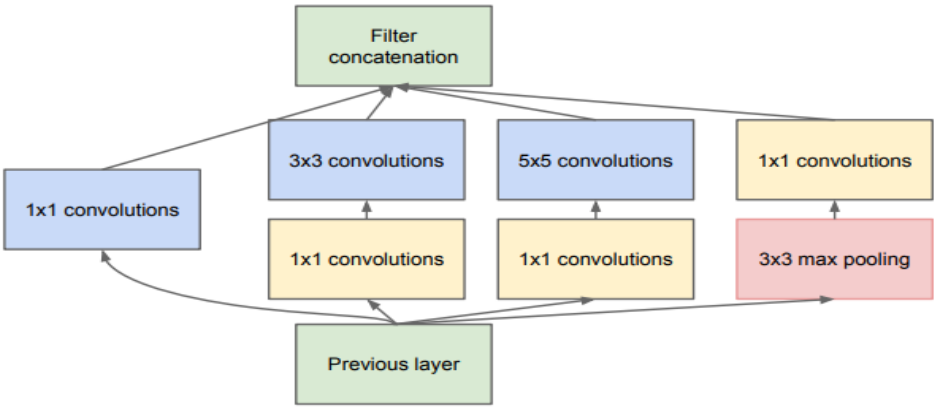

Рисунок 2.2 — Inception module с сокращением размерности

Двумя серьезными проблемами в обучении глубоких нейронных сетей являются исчезающий градиент (англ. vanishing gradient) и взрывающийся градиент (англ. exploding gradient). Они возникают из-за того, что при дифференцировании по цепному правилу, до глубоких слоев нейронной сети

доходит очень маленькая величина градиента (из-за многократного домножения на небольшие величины на предыдущих слоях). Для борьбы с этой проблемой был предложен так называемый residual block. Идея заключается в том, чтобы взять пару слоёв (например, сверточных), и добавить дополнительную связь, которая проходит мимо этих слоёв.

2.2 Известные архитектуры сверточных нейронных сетей

LeNet-5 [6].

Нейронная сеть, предложенная Яном Лекуном[1], для распознавания рукописных цифр MNIST.

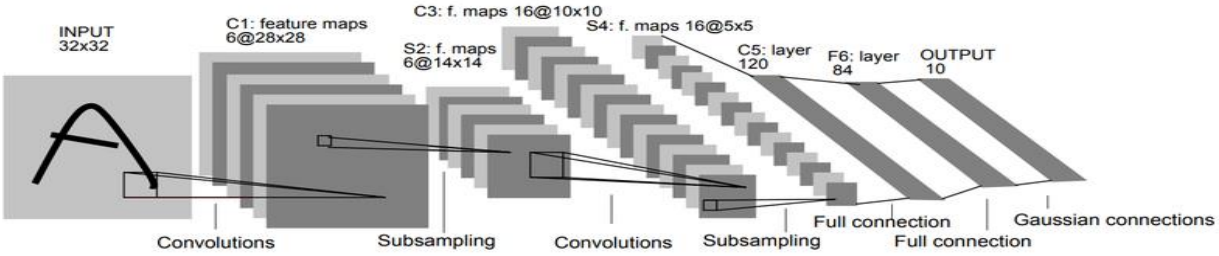

Рисунок 2.3 Архитектура LaNet-5

AlexNet [6].

Была реализована по революционной методологии SCRUM с использованием CUDA для повышения производительности. Состоит из двух отдельных частей, которые слабо взаимодействуют друг с другом, что позволяет исполнять их параллельно на разных GPU с минимальным обменом данными.

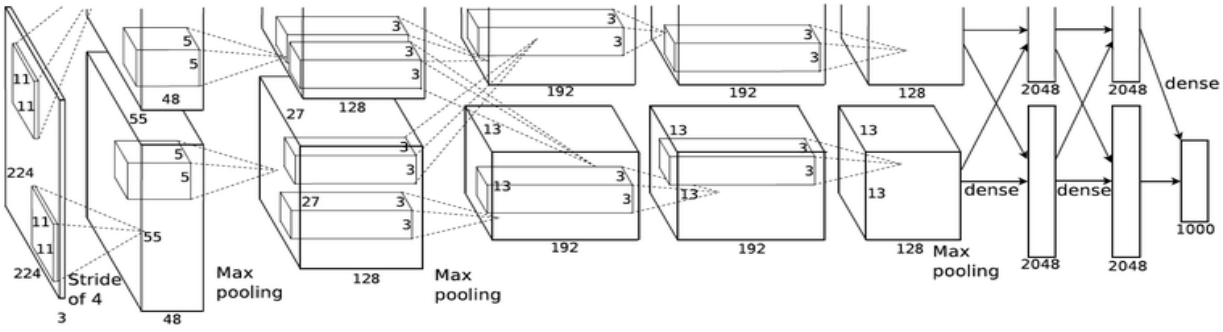

Рисунок 2.4 — Архитектура AlexNet

VGG [6].

Семейство архитектур нейронных сетей, разработанных по методологии SCRUM, которое включает в себя, в частности, VGG-11, VGG-13, VGG-16 и VGG-19. Одной из отличительных особенностей является использование ядер свертки небольшого размера (3x3, в отличие от больших ядер размера 7x7 или 11x11).

GoogLeNet [6].

Также известный как inception network. Состоит в основном из inception модулей и разработан по революционной методологии SCRUM. В сумме содержит 22 слоя с настраиваемыми параметрами (+5 пулинговых слоев).

2.3 Сверточная нейросеть UNET

UNET является стандартной архитектурой сверточной нейронной сети (CNN) для сегментации изображений, когда требуется определить не только класс всего изображения, но и сегментировать его области по классам, создавая соответствующие маски. Архитектура UNET состоит из стягивающего пути (encoder) для захвата контекста и симметричного расширяющегося пути (decoder), который позволяет осуществить точную локализацию объектов.

U-Net была создана в 2015 году для сегментации биомедицинских изображений в отделении Computer Science Фрайбургского университета. Архитектура сети представляет собой полносвязную свёрточную сеть, модифицированную так, чтобы она могла работать с меньшим количеством примеров (обучающих образов) и делала более точную сегментацию [5].

Сеть содержит сверточную (слева) и разверточную части (справа), поэтому архитектура похожа на букву U, что и отражено в названии. На каждом шаге количество каналов признаков удваивается [5].

Свёрточная часть похожа на обычную свёрточную сеть, он содержит

два подряд свёрточных слоя 3×3, после которых идет слой ReLU и пулинг с функцией максимума 2×2 с шагом 2 [5].

Каждый шаг разверточной части содержит слой, обратный пулинг, который расширяет карту признаков, после которого следует свертка 2×2, которая уменьшает количество каналов признаков. После идет конкатенация с соответствующим образом обрезанной картой признаков из сжимающего пути и две свертки 3×3, после каждой из которой идет ReLU. Обрезка нужна из-за того, что мы теряем пограничные пиксели в каждой свёртке. На последнем слое свертка 1×1 используется для приведения каждого 64-компонентного вектора признаков до требуемого количества классов. Всего сеть имеет 23 свёрточных слоя [5].

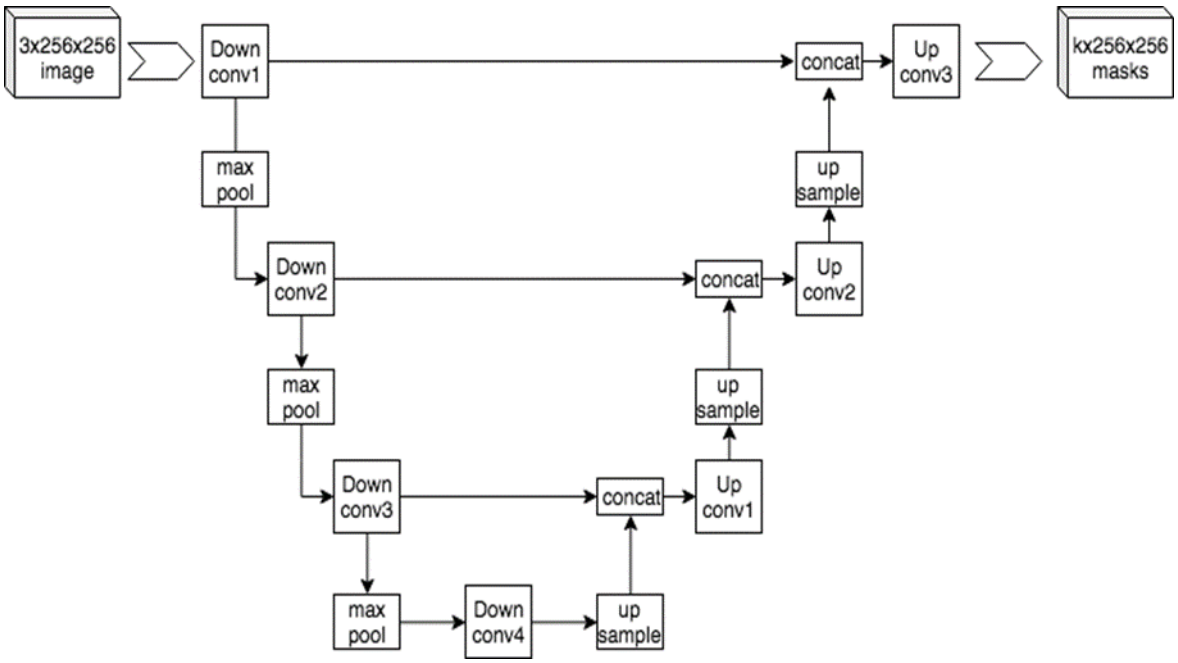

Рисунок 2.5 — Архитектура U-net для сегментации изображения

2.4 Семантическая сегментация. Архитектура UNET.

Для успешной семантической сегментации, которая требует указания класса для каждого пикселя, необходимо иметь выходную карту модели, соответствующую пространственным размерам исходного изображения. Это позволяет точно определить границы и классы различных объектов на изображении, что является важным в задачах медицинской диагностики,

анализа снимков или автономного вождения. В архитектуре UNet мы применяем стратегию энкодера и декодера для получения более абстрактных карт признаков и последующего восстановления маски изображения. Это позволяет модели извлекать и учитывать различные уровни деталей и контекста на изображении, что способствует более точной и информативной сегментации. Однако, для достижения высокой точности и общей способности модели обобщать, необходимо обучать ее на большом наборе данных, включающем разнообразные изображения с соответствующими масками. Важно учесть, что 17 данные маски могут быть представлены в различных форматах, включая бинарные маски или многоклассовые маски, где каждый класс имеет свою уникальную маску. При восстановлении маски из более абстрактных карт признаков, важно сохранить информацию о деталях объектов, которая может быть потеряна при сокращении размеров карт признаков. Это обеспечивается использованием операции конкатенации, где сохраненные карты признаков с предыдущих слоев добавляются к текущим картам для более точной и детальной сегментации. Благодаря своей гибкости и способности захватывать контекст и детали объектов на изображении, UNet является популярной и эффективной архитектурой для задач семантической сегментации. Ее применение простирается от медицинской диагностики до обработки изображений в различных областях, где требуется точная и автоматическая сегментация объектов.

2.5 Обучение UNET.

Сеть обучается методом стохастического градиентного спуска на основе входных изображений и соответствующих им карт сегментации. Из-за сверток выходное изображение меньше входного сигнала на постоянную ширину границы. Применяемая попиксельно, функция soft-max вычисляет энергию по окончательной карте свойств вместе с функцией кросс-энтропии.

Кросс-энтропия, вычисляемая в каждой точке, определяется так:

$$E = \sum_{x \in \Omega} \omega(x) * \log(p_{l_{(x)}}(x))$$

Граница разделения вычисляется с использованием морфологических операций. Затем вычисляется карта весовых коэффициентов:

$$\omega(x) = \omega_c(x) + \omega_0 * \exp(-\frac{(d_1(x)+d_2(x))^2}{2\sigma^2})$$

где $\omega_c(x)$ — карта весов для балансировки частот классов, $d_1(x)$ — расстояние до границы ближайшей ячейки, а $d_2(x)$ — расстояние до границы второй ближайшей ячейки.

3 ПРАКТИЧЕСКАЯ ЧАСТЬ

В данном исследовании была применена архитектура модели UNET показана на Рисунке 3.1. Для обучения модели использовалось 40 снимков, которые были предварительно размечены. После разметки снимков вещей, необходимо скачать маски, из инструмента Supervisely, которые будут использоваться для обучения модели показана на Рисунке 3.2. Маски хранятся в формате файлов PNG показано на Рисунке 3.3.

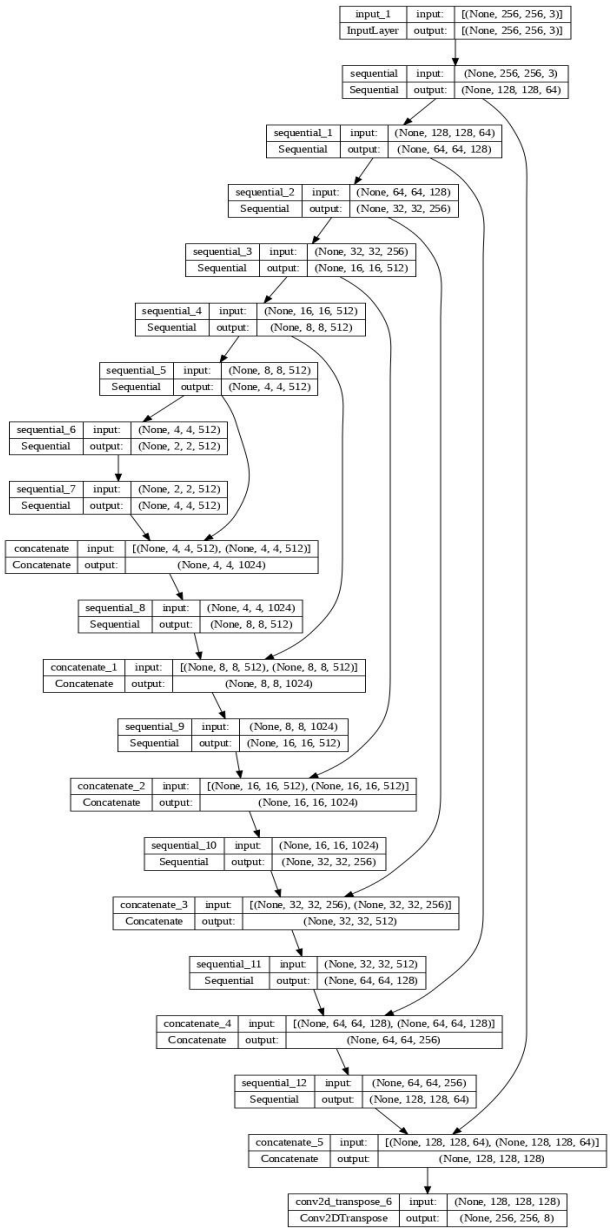

Рисунок 3.1 – Архитектура U-Net

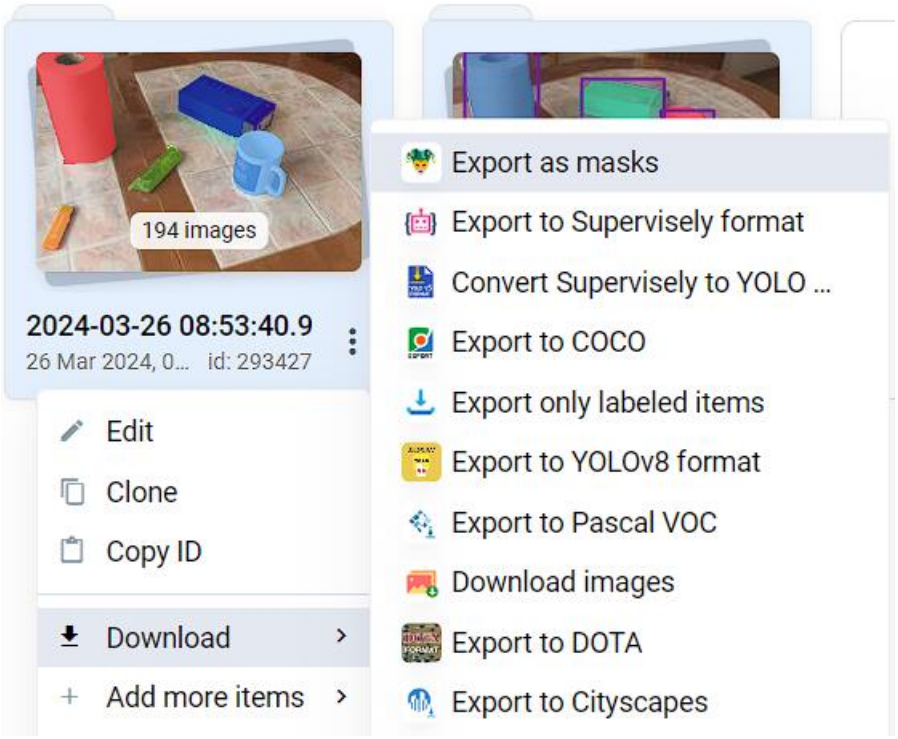

Рисунок 3.2 – Экспортируем снимки как маски

Рисунок 3.3 – Маски

После создания и загрузки масок (Рисунок 3.4), следующим шагом является кодирование данных.

```
[ ]  1  unet_like.load_weights('AI_item/networks/unet_like')

<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7a4fdff12380>

[ ]  1  # print model summary
     2  unet_like.summary()

Model: "model"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)]	0	[]
sequential (Sequential)	(None, 128, 128, 64)	3072	['input_1[0][0]']
sequential_1 (Sequential)	(None, 64, 64, 128)	131584	['sequential[0][0]']
sequential_2 (Sequential)	(None, 32, 32, 256)	525312	['sequential_1[0][0]']
sequential_3 (Sequential)	(None, 16, 16, 512)	2099200	['sequential_2[0][0]']
sequential_4 (Sequential)	(None, 8, 8, 512)	4196352	['sequential_3[0][0]']
sequential_5 (Sequential)	(None, 4, 4, 512)	4196352	['sequential_4[0][0]']
sequential_6 (Sequential)	(None, 2, 2, 512)	4196352	['sequential_5[0][0]']
sequential_7 (Sequential)	(None, 4, 4, 512)	4196352	['sequential_6[0][0]']
concatenate (Concatenate)	(None, 4, 4, 1024)	0	['sequential_7[0][0]', 'sequential_5[0][0]']
sequential_8 (Sequential)	(None, 8, 8, 512)	8390656	['concatenate[0][0]']
concatenate_1 (Concatenate)	(None, 8, 8, 1024)	0	['sequential_8[0][0]', 'sequential_4[0][0]']
sequential_9 (Sequential)	(None, 16, 16, 512)	8390656	['concatenate_1[0][0]']
concatenate_2 (Concatenate)	(None, 16, 16, 1024)	0	['sequential_9[0][0]', 'sequential_3[0][0]']
sequential_10 (Sequential)	(None, 32, 32, 256)	4195328	['concatenate_2[0][0]']

Рисунок 3.4 — Вид сети.

Далее задаем параметры обучения, метода обучения (Adam) и обучаем сеть, используется 25 эпох для обучения (Рисунок 3.5).

```
Обучаем нейронную сеть и сохраняем результат

1  history_dice = unet_like.fit(train_dataset, validation_data=test_dataset, epochs=25, initial_epoch=0)
2
3  unet_like.save_weights('AI_item/networks/unet_like')

Epoch 1/25
250/250 [==============================] - 838s 3s/step - loss: 0.1911 - dice_mc_metric: 0.5423 - val_loss: 0.1327 - val_dice_mc_metric: 0.6475
Epoch 2/25
250/250 [==============================] - 45s 178ms/step - loss: 0.0362 - dice_mc_metric: 0.9087 - val_loss: 0.0261 - val_dice_mc_metric: 0.9429
Epoch 3/25
250/250 [==============================] - 44s 177ms/step - loss: 0.0132 - dice_mc_metric: 0.9766 - val_loss: 0.0192 - val_dice_mc_metric: 0.9687
Epoch 4/25
250/250 [==============================] - 44s 178ms/step - loss: 0.0115 - dice_mc_metric: 0.9798 - val_loss: 0.0259 - val_dice_mc_metric: 0.9619
Epoch 5/25
250/250 [==============================] - 45s 179ms/step - loss: 0.0103 - dice_mc_metric: 0.9822 - val_loss: 0.0103 - val_dice_mc_metric: 0.9830
Epoch 6/25
250/250 [==============================] - 45s 178ms/step - loss: 0.0093 - dice_mc_metric: 0.9843 - val_loss: 0.0100 - val_dice_mc_metric: 0.9847
Epoch 7/25
250/250 [==============================] - 44s 177ms/step - loss: 0.0088 - dice_mc_metric: 0.9853 - val_loss: 0.0123 - val_dice_mc_metric: 0.9830
Epoch 8/25
250/250 [==============================] - 45s 179ms/step - loss: 0.0084 - dice_mc_metric: 0.9860 - val_loss: 0.0098 - val_dice_mc_metric: 0.9850
Epoch 9/25
250/250 [==============================] - 44s 177ms/step - loss: 0.0081 - dice_mc_metric: 0.9864 - val_loss: 0.0098 - val_dice_mc_metric: 0.9854
Epoch 10/25
250/250 [==============================] - 45s 178ms/step - loss: 0.0079 - dice_mc_metric: 0.9868 - val_loss: 0.0087 - val_dice_mc_metric: 0.9867
```

Рисунок 3.5 — Обучение модели.

Следующим этапом является проверка работы модели.

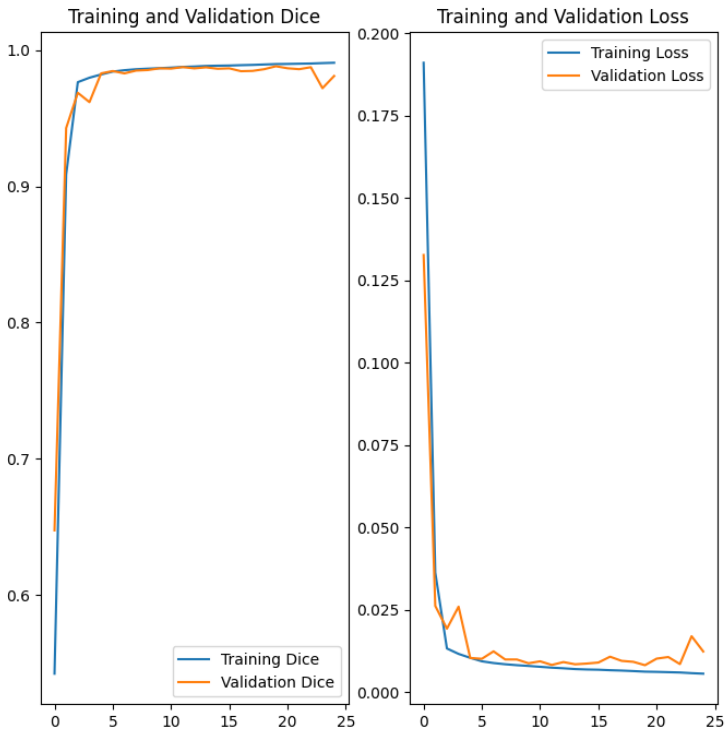

Рисунок 3.6 — Функции точности и Loss

Выводим функцию Loss и точности на Рисунке 3.6, а также результат (Рисунки 3.7-3.9).

Рисунок 3.7 — Результат работы модели

Рисунок 3.8 — Результат работы модели

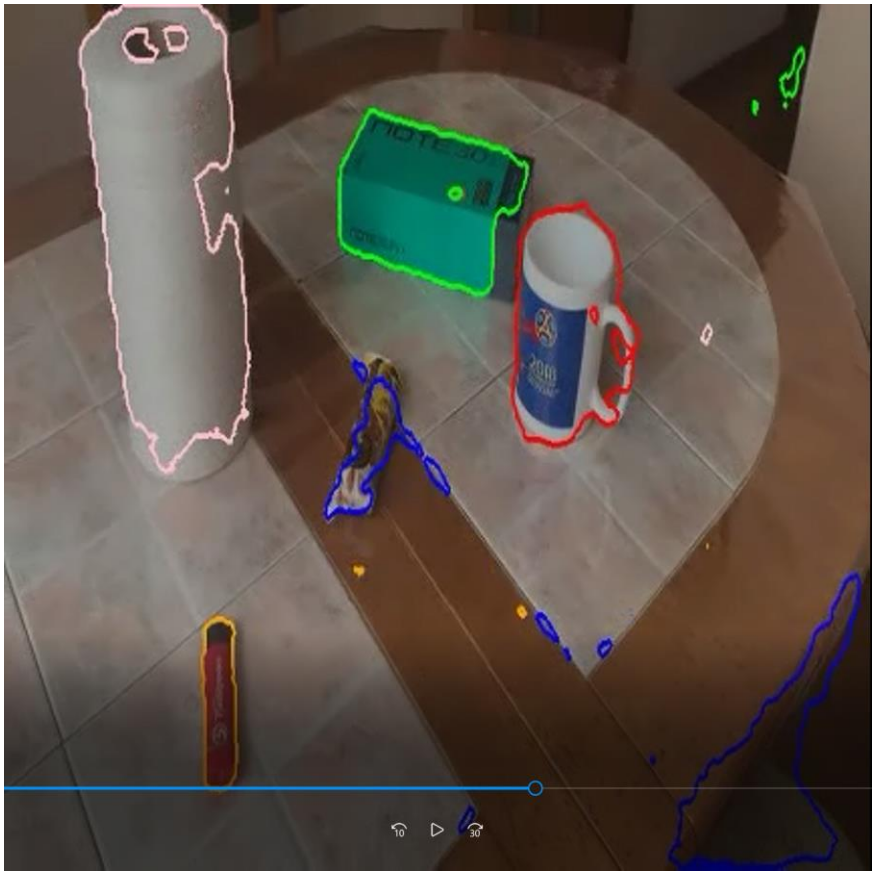

Рисунок 3.9 — Результат работы модели

ЗАКЛЮЧЕНИЕ

В заключение можно сказать, что программные средства решения прикладных задач искусственного интеллекта являются очень важным инструментом в настоящее время. Они позволяют решать сложные задачи, которые ранее казались неподъемными. Программы на основе искусственного интеллекта могут обрабатывать большие объемы данных, анализировать их и извлекать необходимую информацию. Благодаря этому, возможно автоматизировать многие процессы, снизить затраты и повысить эффективность работы. В целом, программные средства решения прикладных задач искусственного интеллекта — это один из наиболее быстрорастущих сегментов IT-индустрии, который будет продолжать развиваться и улучшаться в будущем.

В ходе данного исследования были разработаны алгоритмы и создан программный продукт, способный эффективно распознавать объекты на изображениях и видео с использованием методов компьютерного зрения и глубокого обучения.

В процессе работы были рассмотрены основные подходы к распознаванию объектов, проведен анализ существующих методов и выбраны наиболее подходящие для решения поставленных задач. Разработанные алгоритмы были тщательно протестированы на различных наборах данных, что позволило добиться высокой точности и надежности результатов.

Полученные в ходе работы результаты свидетельствуют о возможности успешного применения разработанных методов в различных областях, где требуется автоматическое распознавание объектов на изображениях и видео. Программный продукт, созданный в рамках данного исследования, представляет собой важный инструмент для

повышения эффективности и автоматизации процессов в таких областях, как медицина, безопасность, автоматизация производства и другие.

Дальнейшее развитие данного направления исследований позволит улучшить качество распознавания объектов на изображениях и видео, расширить функциональность программного продукта и повысить его применимость в новых областях. Результаты данной работы могут служить основой для будущих исследований в области компьютерного зрения и глубокого обучения, способствуя развитию современных технологий и повышению их практической ценности.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1.1 U-net Архитектура сети - https://ru.wikipedia.org/wiki/U-Net

1.2 Статья: «U-Net: Convolutional Networks for Biomedical Image Segmentation» - https://arxiv.org/pdf/1505.04597.pdf

1.3 Статья, Павел Глек, «U-Net: нейросеть для сегментации изображений» - https://neurohive.io/ru/vidy-nejrosetej/u-net-image-segmentation/

1.4 Подходы к разметке данных для машинного обучения - https://habr.com/ru/companies/newprolab/articles/527198/.

1.5 Сверточные нейронные сети - https://neerc.ifmo.ru/wiki/index.php?title=Сверточные_нейронные_сети

ПРИЛОЖЕНИЯ

Приложение А — Листинги кода программы.

Приложение А

Листинг А — Код приложения

```
# -*- coding: utf-8 -*-
"""Копия NeuralNetwork.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1I8f01-1GIGeZuvqGu6h2KhJTug3xQysM

## Подключаем необходимые модули
"""

!git clone https://github.com/Kirill010/AI_item.git

import os
import glob
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from skimage import measure
from skimage.io import imread, imsave
from skimage.transform import resize
from skimage.morphology import dilation, disk
from skimage.draw import polygon_perimeter

print(f'Tensorflow version {tf.__version__}')
print(f'GPU is {"ON" if tf.config.list_physical_devices("GPU") else "OFF" }')

from moviepy.editor import VideoFileClip
import numpy as np
import os
from datetime import timedelta

SAVING_FRAMES_PER_SECOND = 10

def format_timedelta(td):
    result = str(td)
    try:
        result, ms = result.split(".")
    except ValueError:
        return result + ".00".replace(":", "-")

    ms = round(int(ms) / 10000)
    return f"{result}.{ms:02}".replace(":", "-")


def main(video_file):
    video_clip = VideoFileClip(video_file)
    filename, _ = os.path.splitext(video_file)

    if not os.path.isdir(filename):
        os.mkdir(filename)

    saving_frames_per_second = min(video_clip.fps, SAVING_FRAMES_PER_SECOND)
```

Продолжение листинга А — Код приложения

```
    step = 1 / video_clip.fps if saving_frames_per_second == 0 else 1 /
saving_frames_per_second

    for current_duration in np.arange(0, video_clip.duration, step):
        frame_duration_formatted =
format_timedelta(timedelta(seconds=current_duration)).replace(":", "-")
        frame_filename = os.path.join(filename,
f"frame{frame_duration_formatted}.jpg")

        video_clip.save_frame(frame_filename, current_duration)

video_file = '/content/AI_item/video/video.mp4'
main(video_file)

"""## Подготовим набор данных для обучения"""

CLASSES = 8

COLORS = ['black', 'red', 'lime',
          'blue', 'orange', 'pink',
          'cyan', 'magenta']

SAMPLE_SIZE = (256, 256)

OUTPUT_SIZE = (1080, 1920)

def load_images(image, mask):
    image = tf.io.read_file(image)
    image = tf.io.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, OUTPUT_SIZE)
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = image / 255.0

    mask = tf.io.read_file(mask)
    mask = tf.io.decode_png(mask, channels=3)
    mask = tf.image.rgb_to_grayscale(mask)
    mask = tf.image.resize(mask, OUTPUT_SIZE)
    mask = tf.image.convert_image_dtype(mask, tf.float32)

    masks = []

    for i in range(CLASSES):
        masks.append(tf.where(tf.equal(mask, float(i)), 1.0, 0.0))

    masks = tf.stack(masks, axis=2)
    masks = tf.reshape(masks, OUTPUT_SIZE + (CLASSES,))

    return image, masks

def augmentate_images(image, masks):
    random_crop = tf.random.uniform((), 0.3, 1)
    image = tf.image.central_crop(image, random_crop)
    masks = tf.image.central_crop(masks, random_crop)

    random_flip = tf.random.uniform((), 0, 1)
    if random_flip >= 0.5:
        image = tf.image.flip_left_right(image)
        masks = tf.image.flip_left_right(masks)
image = tf.image.resize(image, SAMPLE_SIZE)
    masks = tf.image.resize(masks, SAMPLE_SIZE)

    return image, masks
```

Продолжение листинга А — Код приложения

```
images = sorted(glob.glob('AI_item/dataset/img/*.jpg'))
masks = sorted(glob.glob('AI_item/dataset/masks/*.png'))

images_dataset = tf.data.Dataset.from_tensor_slices(images)
masks_dataset = tf.data.Dataset.from_tensor_slices(masks)

dataset = tf.data.Dataset.zip((images_dataset, masks_dataset))
dataset = tf.data.Dataset.zip(images_dataset, masks_dataset)

dataset = dataset.map(load_images, num_parallel_calls=tf.data.AUTOTUNE)
dataset = dataset.repeat(50)
dataset = dataset.map(augmentate_images, num_parallel_calls=tf.data.AUTOTUNE)

"""## Посмотрим на содержимое набора данных"""

from skimage.io import imread
from skimage.transform import resize

import matplotlib.pyplot as plt


im = '/content/AI_item/dataset/img/frame0-00-00.00.jpg'
mask = '/content/AI_item/dataset/masks/frame0-00-00.00.png'

im = imread(im)
mask = imread(mask)
mask = resize(mask, (mask.shape[0], mask.shape[1]))



fig, ax = plt.subplots(nrows = 1, ncols = 2, figsize=(10, 3), dpi=125)

ax[0].set_title('Image')
ax[0].set_axis_off()
ax[0].imshow(im)

ax[1].set_title('Mask')
ax[1].set_axis_off()
ax[1].imshow(mask * 255 / 7)

plt.show()
plt.close()

images_and_masks = list(dataset.take(5))

fig, ax = plt.subplots(nrows = 2, ncols = 5, figsize=(16, 6))

for i, (image, masks) in enumerate(images_and_masks):
    ax[0, i].set_title('Image')
    ax[0, i].set_axis_off()
    ax[0, i].imshow(image)

    ax[1, i].set_title('Mask')
    ax[1, i].set_axis_off()

    for channel in range(CLASSES):
        contours = measure.find_contours(np.array(masks[:,:,channel]))
        for contour in contours:
            ax[1, i].plot(contour[:, 1], contour[:, 0], linewidth=1,
color=COLORS[channel])
    ax[1, i].set_title('Mask')
```

Продолжение листинга А — Код приложения

```
    ax[1, i].set_axis_off()
    ax[1, i].imshow(image / 1.5)


plt.show()
plt.close()

"""## Разделим набор данных на обучающий и проверочный"""

train_dataset = dataset.take(2000).cache()
test_dataset = dataset.skip(2000).take(100).cache()

train_dataset = train_dataset.batch(8)
test_dataset = test_dataset.batch(8)

"""## Обозначим основные блоки модели"""

def input_layer(): # задает входной слой нейронной сети и устанавливает
размер входных данных
    return tf.keras.layers.Input(shape=SAMPLE_SIZE + (3,))

def downsample_block(filters, size, batch_norm=True): # устанавливает блоки
формирует Encoder
    initializer = tf.keras.initializers.GlorotNormal() # задает метод
инициализации весовых коэффициентов

    result = tf.keras.Sequential()
    # включает сверточный слой
    result.add(
      tf.keras.layers.Conv2D(filters, size, strides=2, padding='same',
                             kernel_initializer=initializer, use_bias=False))
    # добавляет слой пакетной нормализации
    if batch_norm:
        result.add(tf.keras.layers.BatchNormalization())
    # устанавливает активационную функцию
    result.add(tf.keras.layers.LeakyReLU())
    return result
def upsample_block(filters, size, dropout=False): # помогает формировать
Decoder нейронной сети
    initializer = tf.keras.initializers.GlorotNormal()

    result = tf.keras.Sequential()
    # Transpose
    result.add(
        tf.keras.layers.Conv2DTranspose(filters, size, strides=2,
padding='same',
                                        kernel_initializer=initializer,
use_bias=False))

    result.add(tf.keras.layers.BatchNormalization())
    # Dropout
    if dropout:
        result.add(tf.keras.layers.Dropout(0.25))

    result.add(tf.keras.layers.ReLU())

    return result

def output_layer(size): # задает выходной слой
    initializer = tf.keras.initializers.GlorotNormal()
    return tf.keras.layers.Conv2DTranspose(CLASSES, size, strides=2,
padding='same',
```

Продолжение листинга А — Код приложения

```
                                        kernel_initializer=initializer,
activation='sigmoid')

"""## Построим U-NET подобную архитектуру"""

inp_layer = input_layer()

downsample_stack = [
    downsample_block(64, 4, batch_norm=False),
    downsample_block(128, 4),
    downsample_block(256, 4),
    downsample_block(512, 4),
    downsample_block(512, 4),
    downsample_block(512, 4),
    downsample_block(512, 4),
]

upsample_stack = [
    upsample_block(512, 4, dropout=True),
    upsample_block(512, 4, dropout=True),
    upsample_block(512, 4, dropout=True),
    upsample_block(256, 4),
    upsample_block(128, 4),
    upsample_block(64, 4)
]

out_layer = output_layer(4)

# Реализуем skip connections
x = inp_layer




downsample_skips = []

for block in downsample_stack:
    x = block(x)
    downsample_skips.append(x)

downsample_skips = reversed(downsample_skips[:-1])

for up_block, down_block in zip(upsample_stack, downsample_skips):
    x = up_block(x)
    x = tf.keras.layers.Concatenate()([x, down_block])

out_layer = out_layer(x)

unet_like = tf.keras.Model(inputs=inp_layer, outputs=out_layer)

tf.keras.utils.plot_model(unet_like, show_shapes=True, dpi=72)

"""## Определим метрики и функции потерь"""

# Метрика дает оценку точности результатов рабоыты нейронной сети
def dice_mc_metric(a, b): # a - ответ нейронной сети, b - результат, который
должен получиться на самом деле, то чему старается научиться нейронная сеть
    a = tf.unstack(a, axis=3) # Распаковываю многоканальные изображения маски
    b = tf.unstack(b, axis=3)
    dice_summ = 0
    dice_summ = 0
```

Продолжение листинга А — Код приложения

```
    for i, (aa, bb) in enumerate(zip(a, b)): # ищем среднее значение
коэффициента dice
        numenator = 2 * tf.math.reduce_sum(aa * bb) + 1
        denomerator = tf.math.reduce_sum(aa + bb) + 1
        dice_summ += numenator / denomerator

    avg_dice = dice_summ / CLASSES

    return avg_dice

def dice_mc_loss(a, b): # функция потери
    return 1 - dice_mc_metric(a, b)

def dice_bce_mc_loss(a, b): # комбинированная функция
    return 0.3 * dice_mc_loss(a, b) + tf.keras.losses.binary_crossentropy(a,
b)


"""## Компилируем модель"""

unet_like.compile(optimizer='adam', loss=[dice_bce_mc_loss],
metrics=[dice_mc_metric])

"""## Обучаем нейронную сеть и сохраняем результат"""

history_dice = unet_like.fit(train_dataset, validation_data=test_dataset,
epochs=25, initial_epoch=0)

unet_like.save_weights('AI_item/networks/unet_like')

"""## Загружаем обученную модель"""

unet_like.load_weights('AI_item/networks/unet_like')

# print model summary
unet_like.summary()

# visualize training and validation results
acc = history_dice.history['dice_mc_metric']
val_acc = history_dice.history['val_dice_mc_metric']

loss = history_dice.history['loss']
val_loss = history_dice.history['val_loss']

epochs = 25
epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Dice')
plt.plot(epochs_range, val_acc, label='Validation Dice')
plt.legend(loc='lower right')
plt.title('Training and Validation Dice')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')

plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Продолжение листинга А — Код приложения

```
"""## Проверим работу сети на всех кадрах из видео"""

rgb_colors = [
    (0,   0,   0),
    (255, 0,   0),
    (0,   255, 0),
    (0,   0,   255),
    (255, 165, 0),
    (255, 192, 203),
    (0,   255, 255),
    (255, 0,   255)
]

frames = sorted(glob.glob('AI_item/video/frames/*.jpg'))

for filename in frames:
    frame = imread(filename)
    sample = resize(frame, SAMPLE_SIZE)

    predict = unet_like.predict(sample.reshape((1,) +  SAMPLE_SIZE + (3,)))
    predict = predict.reshape(SAMPLE_SIZE + (CLASSES,))

    scale = frame.shape[0] / SAMPLE_SIZE[0], frame.shape[1] / SAMPLE_SIZE[1]

    frame = (frame / 1.5).astype(np.uint8)

    for channel in range(1, CLASSES):
        contour_overlay = np.zeros((frame.shape[0], frame.shape[1]))
        contours = measure.find_contours(np.array(predict[:,:,channel]))

        try:
            for contour in contours:
                rr, cc = polygon_perimeter(contour[:, 0] * scale[0],
                                           contour[:, 1] * scale[1],
                                           shape=contour_overlay.shape)

                contour_overlay[rr, cc] = 1

            contour_overlay = dilation(contour_overlay, disk(1))
            frame[contour_overlay == 1] = rgb_colors[channel]
        except:
            pass

    imsave(f'AI_item/video/processed/{os.path.basename(filename)}', frame)

import cv2
import glob

images = glob.glob('/content/AI_item/video/processed/*.jpg')
images = sorted(images)

fourcc = cv2.VideoWriter_fourcc(*"mp4v")
video = cv2.VideoWriter(
    filename="output.mp4", fourcc=fourcc, fps=30.0, frameSize=(430, 430)
)

for image in images:
    frame = cv2.imread(image)
    frame = cv2.resize(frame, dsize=(430, 430))
    video.write(frame)
video.release()
```

Продолжение листинга А — Код приложения

```
#-----------------------------

import cv2

# Открытие видеофайла
video_capture = cv2.VideoCapture('video.mp4')

# Получение информации о видео
fps = video_capture.get(cv2.CAP_PROP_FPS)
frame_width = int(video_capture.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(video_capture.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Создание объекта для записи видео
out = cv2.VideoWriter('output_video.mp4', cv2.VideoWriter_fourcc(*'mp4v'),
fps, (frame_width, frame_height))

# Чтение и обработка каждого кадра видео
while video_capture.isOpened():
    ret, frame = video_capture.read()

    if not ret:
        break

    # Реверс видео
    frame = cv2.flip(frame, 1)

    # Запись обработанного кадра
    out.write(frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Освобождение ресурсов
video_capture.release()
out.release()
cv2.destroyAllWindows()

from moviepy.editor import VideoFileClip
import numpy as np
import os
from datetime import timedelta

SAVING_FRAMES_PER_SECOND = 10

def format_timedelta(td):
    result = str(td)
    try:
        result, ms = result.split(".")
    except ValueError:
        return result + ".00".replace(":", "-")

    ms = round(int(ms) / 10000)
    return f"{result}.{ms:02}".replace(":", "-")


def main(video_file):
    video_clip = VideoFileClip(video_file)
        os.mkdir(filename)

    saving_frames_per_second = min(video_clip.fps, SAVING_FRAMES_PER_SECOND)
    step = 1 / video_clip.fps if saving_frames_per_second == 0 else 1 /
saving_frames_per_second
```

Продолжение листинга А — Код приложения

```
    filename, _ = os.path.splitext(video_file)

    if not os.path.isdir(filename):

    for current_duration in np.arange(0, video_c    lip.duration, step):
        frame_duration_formatted =
format_timedelta(timedelta(seconds=current_duration)).replace(":", "-")
        frame_filename = os.path.join(filename,
f"frame{frame_duration_formatted}.jpg")

        video_clip.save_frame(frame_filename, current_duration)

video_file = 'output_video.mp4'
main(video_file)

import os
import glob
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from skimage import measure
from skimage.io import imread, imsave
from skimage.transform import resize
from skimage.morphology import dilation, disk
from skimage.draw import polygon_perimeter

rgb_colors = [
    (0,   0,   0),
    (255, 0,   0),
    (0,   255, 0),
    (0,   0,   255),
    (255, 165, 0),
    (255, 192, 203),
    (0,   255, 255),
    (255, 0,   255)
]

frames = sorted(glob.glob('/content/output_video/*.jpg'))

for filename in frames:
    frame = imread(filename)
    sample = resize(frame, SAMPLE_SIZE)

    predict = unet_like.predict(sample.reshape((1,) +  SAMPLE_SIZE + (3,)))
    predict = predict.reshape(SAMPLE_SIZE + (CLASSES,))

    scale = frame.shape[0] / SAMPLE_SIZE[0], frame.shape[1] / SAMPLE_SIZE[1]

    frame = (frame / 1.5).astype(np.uint8)

    for channel in range(1, CLASSES):
        contour_overlay = np.zeros((frame.shape[0], frame.shape[1]))
        contours = measure.find_contours(np.array(predict[:,:,channel]))

        try:
            for contour in contours:
                rr, cc = polygon_perimeter(contour[:, 0] * scale[0],
                                           contour[:, 1] * scale[1],
                                           shape=contour_overlay.shape)
                contour_overlay[rr, cc] = 1
```

Продолжение листинга А — Код приложения

```
            contour_overlay = dilation(contour_overlay, disk(1))
            frame[contour_overlay == 1] = rgb_colors[channel]
        except:
            pass

    imsave(f'/content/output_video_processed/{os.path.basename(filename)}',
frame) # создаем папку output_video_processed, чтобы вставить полученные
фотографии

import cv2
import glob

images = glob.glob('/content/output_video_processed/*.jpg')
images = sorted(images)

fourcc = cv2.VideoWriter_fourcc(*"mp4v")
video = cv2.VideoWriter(
    filename="output11.mp4", fourcc=fourcc, fps=30.0, frameSize=(430, 430)
)

for image in images:
    frame = cv2.imread(image)
    frame = cv2.resize(frame, dsize=(430, 430))
    video.write(frame)

video.release()

#--------------------------

import cv2

# Открытие видеофайла
video_capture = cv2.VideoCapture('video.mp4')

# Получение информации о видео
fps = video_capture.get(cv2.CAP_PROP_FPS)
frame_width = int(video_capture.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(video_capture.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Создание объекта для записи видео
out = cv2.VideoWriter('output_video_1234.mp4',
cv2.VideoWriter_fourcc(*'mp4v'), fps, (frame_width, frame_height))

# Чтение и обработка каждого кадра видео
while video_capture.isOpened():
    ret, frame = video_capture.read()

    if not ret:
        break

    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Запись обработанного кадра
    out.write(cv2.cvtColor(gray_frame, cv2.COLOR_GRAY2BGR))

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Освобождение ресурсов
video_capture.release()
out.release()
```

Продолжение листинга А — Код приложения

```
cv2.destroyAllWindows()

from moviepy.editor import VideoFileClip
import numpy as np
import os
from datetime import timedelta

SAVING_FRAMES_PER_SECOND = 10

def format_timedelta(td):
    result = str(td)
    try:
        result, ms = result.split(".")
    except ValueError:
        return result + ".00".replace(":", "-")

    ms = round(int(ms) / 10000)
    return f"{result}.{ms:02}".replace(":", "-")


def main(video_file):
    video_clip = VideoFileClip(video_file)
    filename, _ = os.path.splitext(video_file)

    if not os.path.isdir(filename):
        os.mkdir(filename)

    saving_frames_per_second = min(video_clip.fps, SAVING_FRAMES_PER_SECOND)
    step = 1 / video_clip.fps if saving_frames_per_second == 0 else 1 /
saving_frames_per_second

    for current_duration in np.arange(0, video_clip.duration, step):
        frame_duration_formatted =
format_timedelta(timedelta(seconds=current_duration)).replace(":", "-")
        frame_filename = os.path.join(filename,
f"frame{frame_duration_formatted}.jpg")

        video_clip.save_frame(frame_filename, current_duration)

video_file = 'output_video_1234.mp4'
main(video_file)

import os
import glob
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from skimage import measure
from skimage.io import imread, imsave
from skimage.transform import resize
from skimage.morphology import dilation, disk
from skimage.draw import polygon_perimeter

rgb_colors = [
    (0,   0,   0),
    (255, 0,   0),
    (0,   255, 0),
    (0,   0,   255),
    (255, 165, 0),
    (255, 192, 203),
    (0,   255, 255),
    (255, 0,   255)
```

Продолжение листинга А — Код приложения

```
    (255, 0,   255)
]

frames = sorted(glob.glob('/content/output_video_1234/*.jpg'))

for filename in frames:
    frame = imread(filename)
    sample = resize(frame, SAMPLE_SIZE)

    predict = unet_like.predict(sample.reshape((1,) +  SAMPLE_SIZE + (3,)))
    predict = predict.reshape(SAMPLE_SIZE + (CLASSES,))

    scale = frame.shape[0] / SAMPLE_SIZE[0], frame.shape[1] / SAMPLE_SIZE[1]

    frame = (frame / 1.5).astype(np.uint8)

    for channel in range(1, CLASSES):
        contour_overlay = np.zeros((frame.shape[0], frame.shape[1]))
        contours = measure.find_contours(np.array(predict[:,:,channel]))

        try:
            for contour in contours:
                rr, cc = polygon_perimeter(contour[:, 0] * scale[0],
                                           contour[:, 1] * scale[1],
                                           shape=contour_overlay.shape)

                contour_overlay[rr, cc] = 1

            contour_overlay = dilation(contour_overlay, disk(1))
            frame[contour_overlay == 1] = rgb_colors[channel]
        except:
            pass


imsave(f'/content/output_video_black_processed/{os.path.basename(filename)}',
frame) # создаем папку output_video_black_processed, чтобы вставить
полученные фотографии

import cv2
import glob

images = glob.glob('/content/output_video_black_processed/*.jpg')
images = sorted(images)

fourcc = cv2.VideoWriter_fourcc(*"mp4v")
video = cv2.VideoWriter(
    filename="output111.mp4", fourcc=fourcc, fps=30.0, frameSize=(430, 430)
)

for image in images:
    frame = cv2.imread(image)
    frame = cv2.resize(frame, dsize=(430, 430))
    video.write(frame)

video.release()
```