Санкт-Петербургский государственный университет

Пашкин Илья Андреевич

Выпускная квалификационная работа

Применение компьютерного зрения для изменения дизайна комнаты на фото

Уровень образования: бакалавриат

Направление: 01.03.02 «Прикладная математика и информатика»

Основная образовательная программа СВ.5005.2017 «Прикладная математика, фундаментальная информатика и программирование»

Научный руководитель:

Кандидат физико-математических наук, доцент

Козынченко Владимир Александрович

Рецензент:

Доктор физико-математических наук, профессор

Андрианов Сергей Николаевич

Санкт-Петербург

2021 г.

Содержание

Введение 3

Постановка задачи 5

Обзор литературы 6

Глава 1. Анализ датасета 7

Глава 2. Немного о U-Net 9

Глава 3. Решение задачи сегментации 11

Глава 4. Решение задачи перекраски 21

Выводы 26

Заключение 27

Список использованных источников 28

Приложение 1 29

Приложение 2 30

Введение

В наше время, когда активно развиваются нейронные сети, появляется возможность решать множество задач, точный алгоритм решения которых человеку придумать сложно, он состоял бы из огромного числа пунктов, которые пришлось бы прописывать вручную.

В частности, с помощью нейронных сетей решаются задачи компьютерного зрения, т.е. задачи, подразумевающие программную обработку изображений или видео для выдачи какого-либо результата (например, задачи классификации, кластеризации, сегментации, восстановления от воздействия шума).

В данной работе решалась одна из таких задач, суть которой состояла в следующем: пользователь загружает фотографию своей комнаты в программу, выбирает любой желаемый цвет, после чего программа перекрашивает в этот цвет обои на фотографии.

Такая программа будет полезна тем, кто собирается менять интерьер у себя в комнате: не придется несколько раз ходить в магазин, чтобы подобрать нужный цвет. Для этого достаточно осмотреть измененную комнату в программе и купить то, что нужно, сразу. Также эта программа выгодна и магазину по продаже настенных покрытий, ее наличие является конкурентным преимуществом.

Среди реализованных на данный момент аналогов стоит выделить мобильное приложение для Android, Dulux Visualizer, которое во время перемещения камеры телефона меняет цвет стен в режиме реального времени. Необходимость пользователя указывать стену вручную для работы программы является, одновременно, и недостатком, и достоинством программы. Без указания пользователя программа не может понять, какие оттенки цвета считаются стеной, а если на фотографии слишком много разделенных объектами участков стены, нужно указывать на каждый из них. С другой стороны, в приложении есть возможность перекрасить одну стену комнаты одним цветом, а соседнюю стену – другим.

Также стоит отметить мобильное приложение Paint Tester, оно выполняет похожую задачу, однако не производит сегментацию, оно перекрашивает указанные участки стены по принципу, похожему на принцип действия инструмента «Волшебная палочка», который доступен, например, в приложениях Adobe Photoshop и Paint.NET, из-за чего часть пикселов не закрашивается в нужный цвет.

В главе 1 данной работы описан выбор фотографий из найденного датасета для обучения модели сегментации стен. В главе 2 кратко описана структура нейронных сетей вида U-Net и принцип их работы. В главе 3 подробно описан процесс обучения моделей для решения задачи сегментации, описано их тестирование, действия по повышению качества работы обучения моделей, а также по повышению качества получаемого результата. В главе 4 описан процесс решения задачи перекраски изображения: первоначальное решение, а затем, его модификация.

Постановка задачи:

Разработать прикладное решение, которое по фотографии комнаты позволяет заменить цвет настенного покрытия на любой другой. Для этого необходимо решить следующие алгоритмические задачи:

1) Решить задачу семантической сегментации стен на изображении, т.е.:
   a) Найти датасет с разнообразными фотографиями комнат, на которых в отдельных изображениях-масках стены выделены одним и тем же цветом, проверить все фотографии на правильность сегментации, составить обучающую и валидационную выборки из тех, которые наиболее подходят;
   b) Построить необходимую для задачи модель нейронной сети, т.е. модель, которая получает на вход изображение, выделяет нужные признаки, и в итоге, преобразует его в изображение-маску того же размера, что и исходное;
   c) Подбирая параметры сети, обучить модель как можно точнее сегментировать стены.
2) Разработать программу, которая меняет цвет сегментированной части на цвет, заданный пользователем. При этом изображение комнаты должно выглядеть реалистично. Для этого необходимо перекрашивать выделенную нейронной сетью маску с учетом освещенности (например, максимального значения цветовых компонент каждого пиксела) этой области на исходном изображении, т.е. затемнять новое изображение в теневых областях стен, в углах комнаты.

Для выполнения поставленной задачи в данной работе был использован язык программирования Python 3.8, с использованием библиотек:

- Keras 2.4.3;
- Numpy 1.20.1;
- Matplotlib 3.3.4;
- Scikit-image 0.18.1.

В качестве среды разработки использовалась PyCharm.

Обзор литературы

В книге А.Ю. Тропченко, А.А. Тропченко «Методы вторичной обработки и распознавания изображений» [2] можно найти информацию о различных методах и алгоритмах сегментации изображений, а также условия, на которые стоит обратить внимание при выборе определенного метода.

Принцип работы сверточной нейронной сети вида U-Net описан в книге S. Pattanayak «Pro Deep Learning with TensorFlow» [9]. Этот вид архитектуры был выбран для решения поставленной задачи.

В книге A. Gulli, S. Pal «Deep Learning with Keras» [3] подробно описано, как с помощью библиотеки Keras можно реализовывать работу нейронных сетей для различных задач.

Подробную информацию с примерами использования широко используемой библиотеки Numpy можно узнать из книги P. Deitel, H. Deitel «Python for Programmers: with Big Data and Artificial Intelligence Case Studies» [10], также в книге показано применение библиотеки Matplotlib для визуализации данных.

Глава 1. Анализ датасета

Сначала был найден датасет [7], в котором содержались 1800 файлов, в каждом из них: трехмерный тензор, описывающий пикселы фотографии комнаты, матрица, описывающая соответствующее сегментированное изображение (черно-белая картинка, на которой каждый оттенок обозначает один из 23 классов различных объектов и поверхностей), а также – по 2 таблицы с числами, описывающие глубину изображения (расстояния до объектов на фотографии, не нужны для поставленной задачи).

Ниже, на рис. 1, приведены примеры фотографий (сегментированные, на самом деле, состоят из оттенков серого, но при выводе их в виде графика с помощью библиотеки Matplotlib [6], выводятся цветными для наглядности).

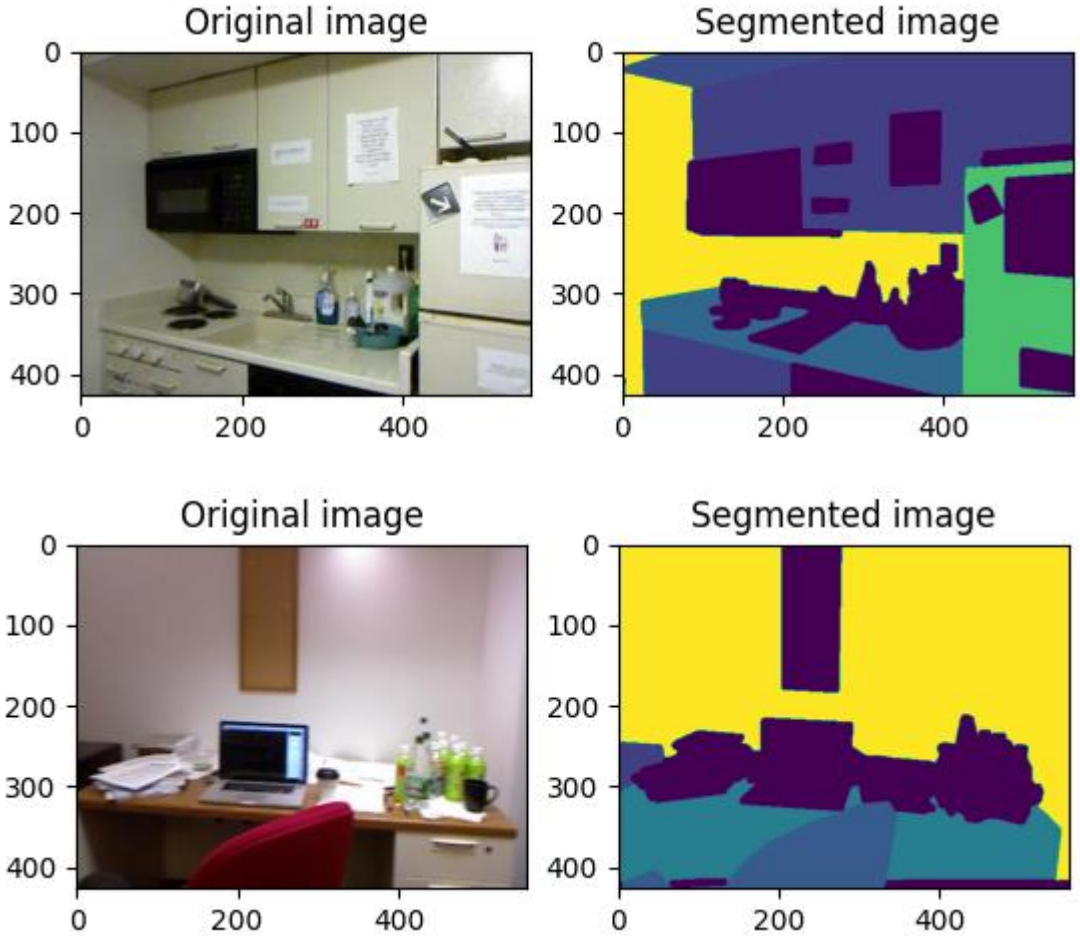

Рис. 1 – Визуализация тензоров датасета в виде картинок с помощью библиотеки Matplotlib

После того, как из датасета тензорные данные были преобразованы к формату изображения с помощью библиотеки Scikit-image [8], все пикселы черно-белых изображений-масок, которые состояли из 23 различных оттенков серого были изменены под поставленную задачу, т.е. любой пиксел, не относящийся к цвету, которым обозначались стены, менялся на черный цвет, остальные же, т.е. искомые, менялись на светло-серый. Таким образом, все маски стали выглядеть, как показано на рис. 2:

Рис. 2 – Преобразование масок к бинарным изображениям

В ходе работы после приведения масок к необходимому виду, выяснилось, что на масках, как и на изображениях, которые были извлечены из файлов датасета, оказались шумы, которые проявлялись в волнообразности переходов от одних классов объектов к другим, что делало дальнейшую реализацию решения задачи сегментации невозможной. Однако проблема оказалась в преобразовании тензоров в изображения формата JPEG, при котором происходило сжатие и создавались шумы. Эта проблема разрешилась путем преобразования к формату PNG.

Затем, были исключены из датасета те фотографии, на масках которых стены были отображены неправильно, т.е. значительная часть стены не была отмечена, либо – наоборот, отмечено было много лишнего как стена.

Глава 2. Немного о U-Net

После того, как была подготовлена обучающая выборка, далее была построена модель нейронной сети с U-Net-подобной архитектурой.

U-Net представляет из себя своеобразную сверточную нейронную сеть (рис. 3). Впервые она была создана в 2015-м году для сегментации биомедицинских изображений.

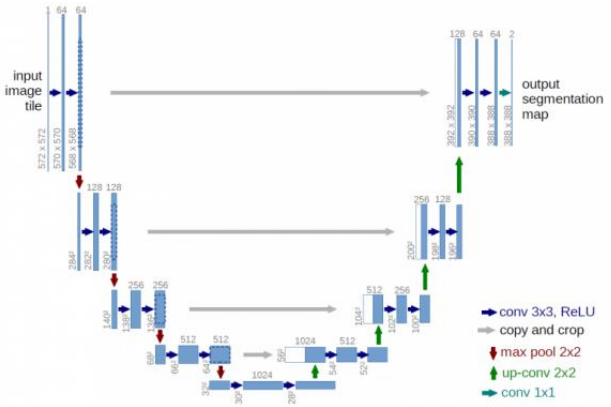

Рис. 3 – U-Net [1]

Эта архитектура представляет из себя последовательность сверточных слоев, чередующихся со слоями MaxPooling. Для задачи классификации изображений могла бы подойти обычная сверточная нейронная сеть, состоящая из последовательности слоев Convolutional и MaxPooling, после которых расположены полносвязные слои, но т.к. результатом семантической сегментации является не просто метка класса, или границы ограничивающего искомый объект прямоугольника (как, например, в задаче детекции), а полноценное изображение высокого разрешения, в котором каждый пиксел относится к какому-либо отдельному классу, значит, качество выходного изображения играет важную роль.

Т.е. необходимо сжатое изображение привести к исходному размеру. Для этого в U-Net далее идет операция обратной (транспонированной) свертки, в которой на каждом этапе происходит расширение сжатого изображения.

После всех этих операций, на выходе нейронной сети формируется маска изображения.

В программе также на каждом этапе свертки и обратной (транспонированной) свертки используются слои Dropout [4], которые каждую итерацию отключают некоторый процент случайно выбранных нейронов, для того, чтобы избежать переобучения (это явление, при котором модель показывает хорошие результаты на примерах из обучающей выборки, но

на любых других примерах – значительно хуже, т.е. у модели не развита способность к обобщению).

Таким образом, на вход программе подается выборка изображений, которые преобразуются к одному размеру, а затем поступают на вход нейронной сети, и происходит обучение.

Глава 3. Решение задачи сегментации

Сначала был произведен процесс обучения 1-й модели в течение 50 эпох с выборкой, размером в 1400 изображений (90% из них были включены в обучающую выборку, 10% – валидационную).

Ниже представлены графики зависимости точности и потерь (на обучающей выборке и на валидационной) от количества пройденных эпох обучения (рис. 4).

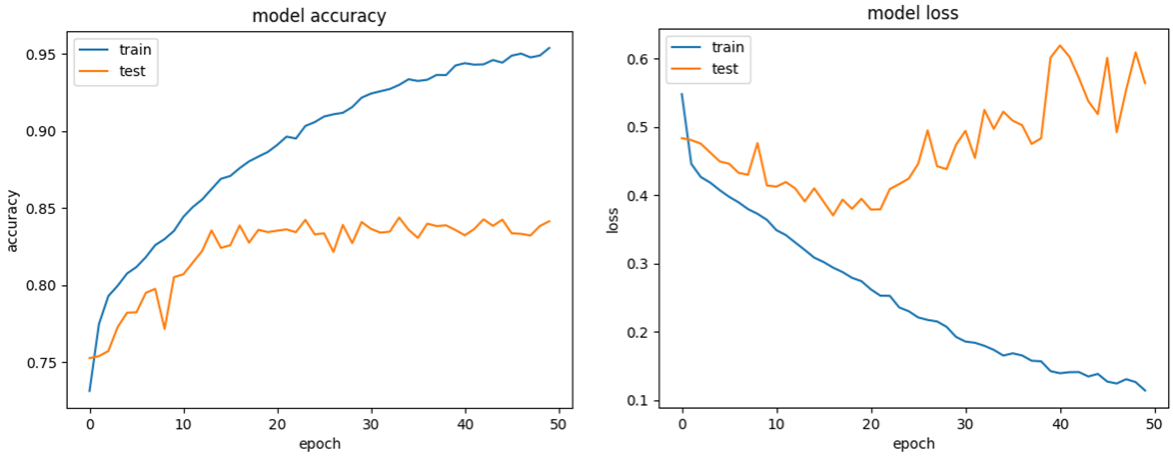

Рис. 4 – Графики точности и потерь при обучении 1-й модели

Значения точности на каждой эпохе вычисляется как отношение суммарного количества пикселов фотографий данной эпохи, которые были правильно классифицированы нейронной сетью, к количеству всех пикселов этих фотографий.

Функцией потерь в данной работе является “Binary cross-entropy” (бинарная логарифмическая функция), значение которой на каждой эпохе вычисляется по формуле:

$$Loss = -\frac{1}{n} \times \sum_{i=1}^{n} (y_i \times log\, \hat{y}_i + (1 - y_i) \times log(1 - \hat{y}_i), \quad (1)$$

где n – размер выходного слоя модели (количество скалярных значений в нем), $\hat{y}_i$ – i-е скалярное значение выходного слоя модели, а y_i – соответствующее целевое значение.

После обучения 1-й модели было дано отдельное входное изображение [5], не входящее в датасет, которое она должна была распознать. Результат представлен на рис. 5:

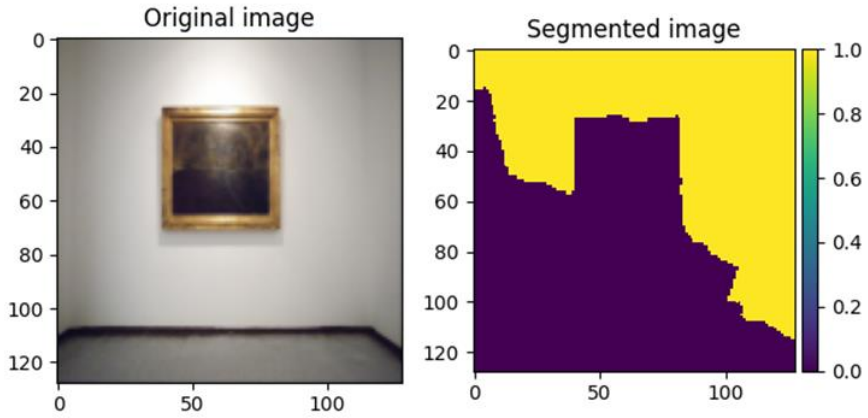

Рис. 5 – Результат работы 1-й обученной модели

Сеть обучилась недостаточно хорошо из-за того, что часть изображений была размечена неправильно в сложных областях.

После этого, фотографии выбирались более тщательно: убирались из выборки либо фотографии с неверной сегментацией, либо те, на которых изображены слишком сложные комнаты (на некоторых человеку сложно распознать стены).

Для обучения второй модели, в общую выборку было включено 1000 фотографий. Графики зависимости точности и потерь от количества эпох представлены ниже, на рис. 6.

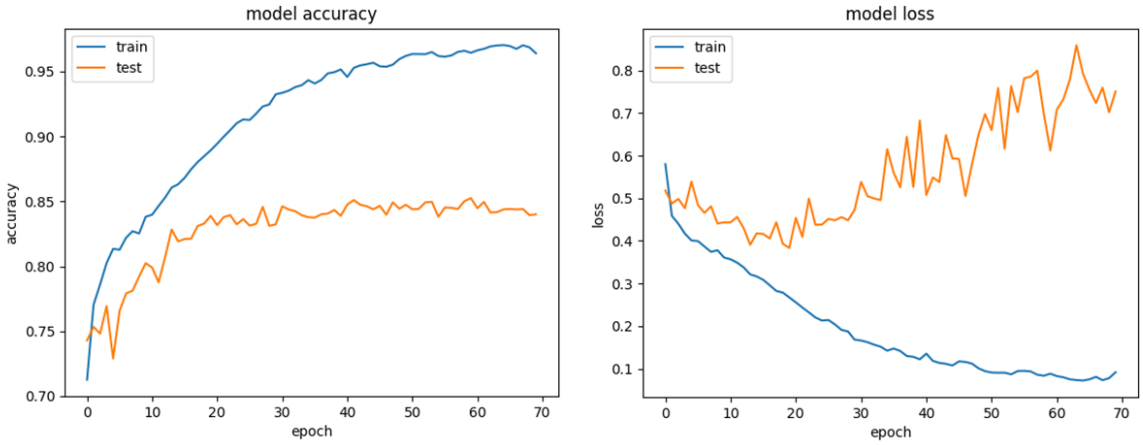

Рис. 6 – Графики точности и потерь при обучении 2-й модели

Обучение на новой выборке происходило в течение 70 эпох, хотя уже после 20-й точность на валидационной выборке колебалась от 83% до 85%. Учитывая факт, что на обучающей выборке на протяжении всех эпох точность монотонно росла, и достигла примерно 96%, можно сказать, что нейронная сеть переобучилась, что, также, произошло и в первом случае. Однако, при проверке 2-й модели на той же фотографии, было замечено,

что программа значительно лучше распознала стены, хотя и с некоторыми погрешностями, особенно, по краям (см. рис. 7).

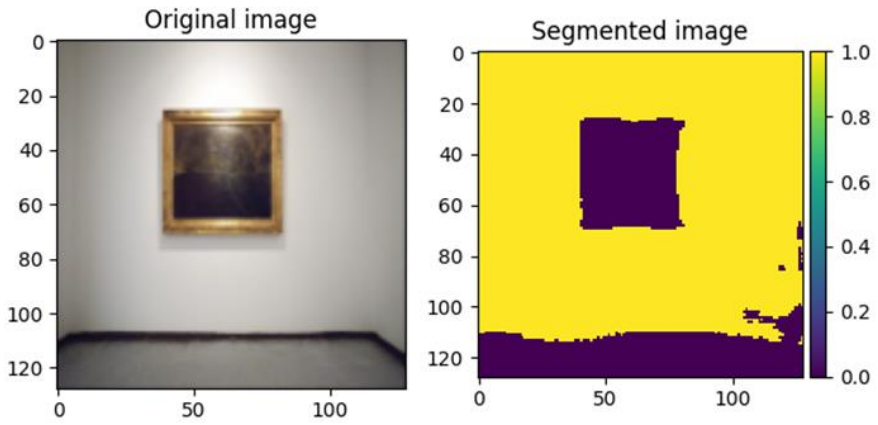

Рис. 7 – Результат работы 2-й обученной модели

Далее было решено исключить из выборки фотографии, на которых при проверке были обнаружены небольшие дефекты, например, как на рис. 8:

Рис. 8 – Пример неправильных входных данных при обучении второй модели: искомый класс не содержит значительную часть стены

Здесь не отмечены области между ступеньками лестницы, что значительно сказывается на обучении.

На рис. 9 представлен пример фотографии, маска которой почти не содержит меток класса, соответствующих стене, и при этом, шкаф во многих местах является достаточно однородной поверхностью, что усложняет обучение нейронной сети.

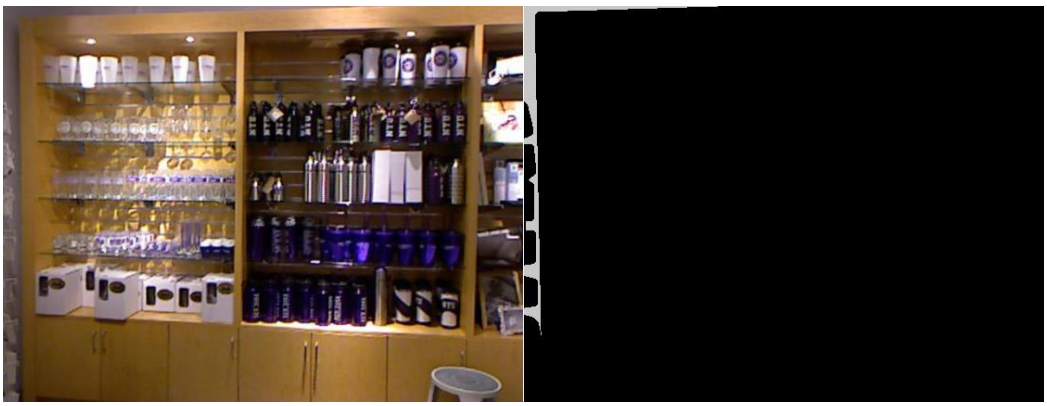

Рис. 9 – Пример сложных входных данных, стены почти отсутствуют на фотографии

Третья выборка состояла уже из 640 фотографий, 64 из которых были включены в валидационный набор.

Новая (3-я) модель обучалась в течение 70 эпох, с теми же параметрами обучения. Из графика (рис. 10) видно, что точность не сильно изменилась.

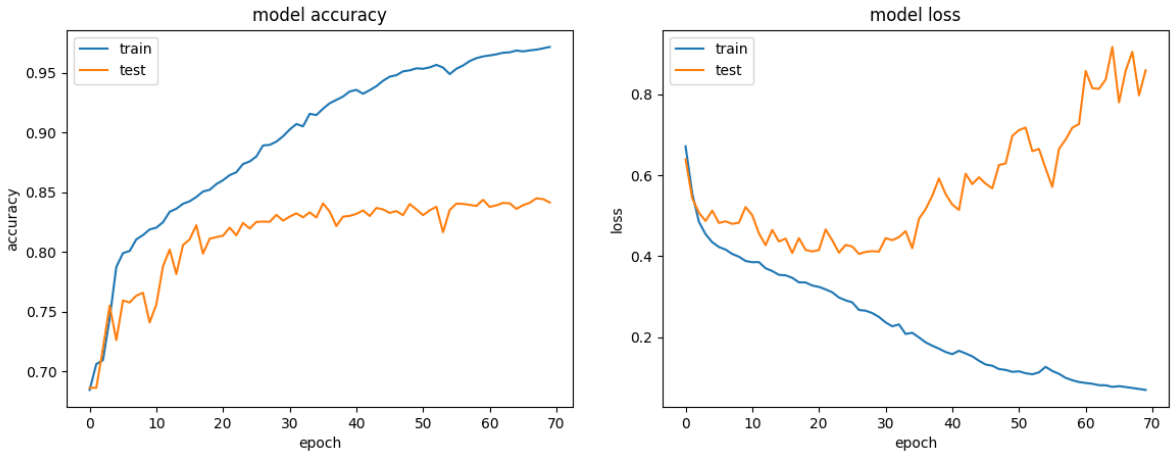

Рис. 10 – Графики точности и потерь при обучении 3-й модели

Результат работы 3-й модели (см. рис. 11) выглядит немного хуже, хотя фотографии из новой выборки являются более правильными (возможно, нехватка обучающих изображений).

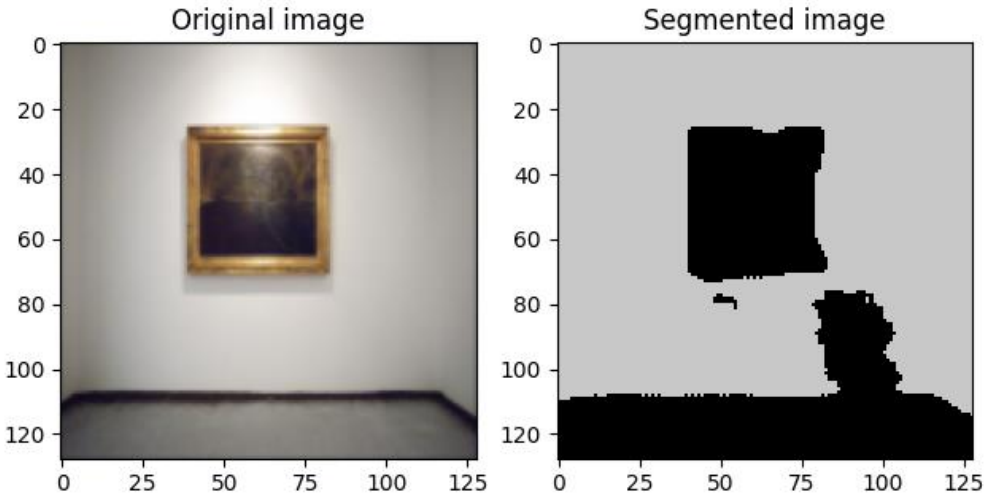

Рис. 11 – Результат работы 3-й обученной модели

После этого, для того чтобы устранить возможную проблему в нехватке обучающих данных, была использована аугментация, т.е. каждый раз перед тем, как фотография и маска подавались на вход нейросети, они были деформированы специальным образом.

Параметры аугментации представлены ниже в табл. 3.1:

Таблица 3.1 – Параметры аугментации

Параметр аугментации	Значение	Пояснение
Приближение	От 0.65 до 1	Преобразованное изображение является приближением исходного в случайной его области с размерами от 0.65 до 1 (без изменений) от исходного
Поворот	5 гр.	Поворот изображения на случайное число градусов от 0 до 5 в одну из двух сторон
Яркость	От 0.6 до 1.2	Яркость изображения варьируется от 0.6 до 1.2 от яркости исходного
Горизонтальный поворот	Да	Изображение может преобразовываться к зеркальному отображению исходного относительно вертикальной оси

После обучения с теми же параметрами, но с использованием аугментации, графики точности и потерь были следующими (см. рис. 12):

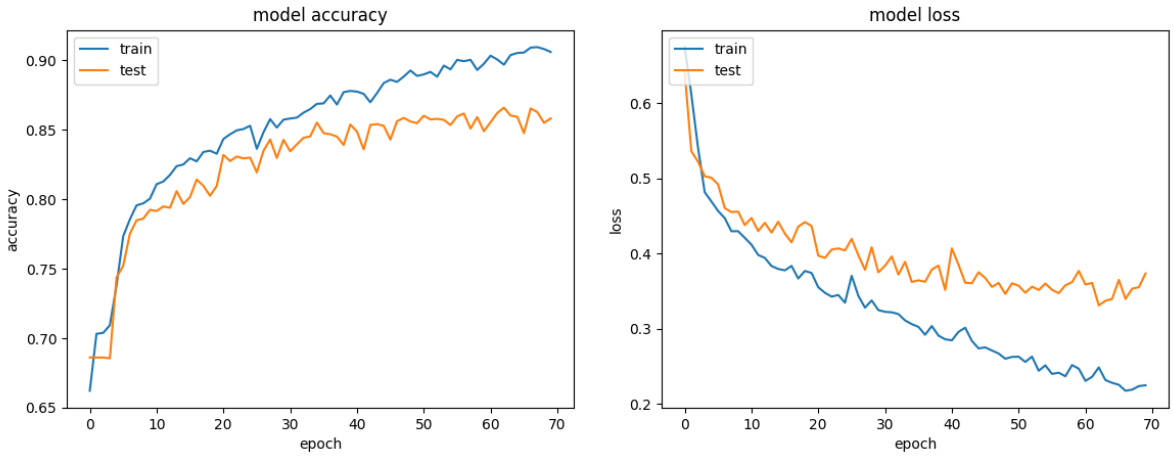

Рис. 12 – Графики точности и потерь при обучении 4-й модели

После обучения точность на валидационном наборе данных поднялась выше 85%, а на той же фотографии модель отработала значительно лучше, чем предыдущая. Недостаточно точными были лишь граничные точки (см. рис. 13).

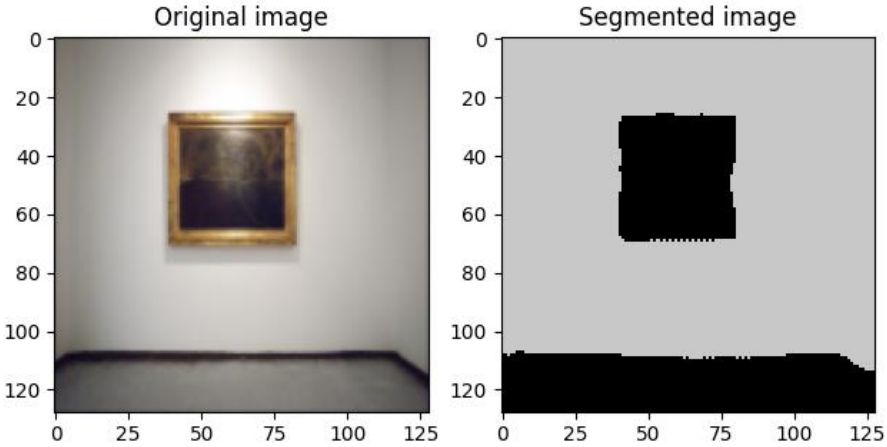

Рис. 13 – Результат работы 4-й обученной модели

Подробно архитектура моделей 1-4 представлена в табл. Б.1 приложения Б.

Как было упомянуто ранее, фотографии преобразуются к одному и тому же размеру перед тем, как они подаются на вход нейросети. Все рассмотренные выше модели получали на вход фотографии разрешения 128x128, которое было и у масок на выходе. Такое разрешение является слишком низким, как для пользователя, который желает увидеть комнату в новом образе, так и для самой нейросети, которая обучается на фотографиях и масках такой размерности. Примеры недостатков такого преобразования представлены на рис. 14:

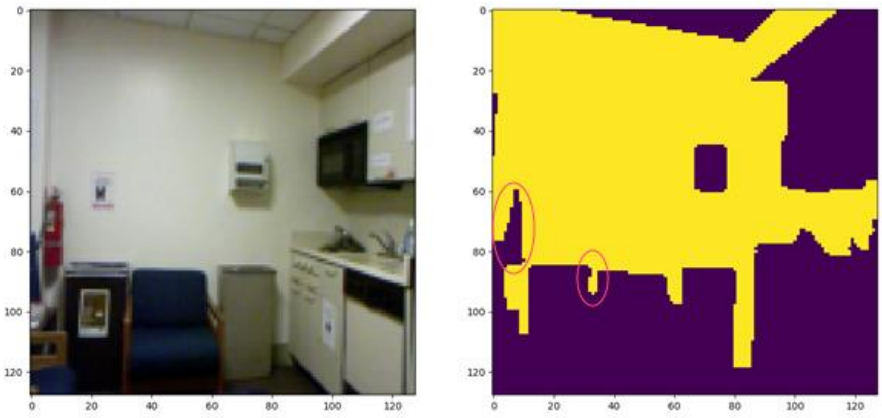

Рис. 14 – Пример фотографии и маски, преобразованных к размерности 128x128

Здесь видно, что один узкий участок стены становится шире при преобразовании, а огнетушитель, например, сильно деформируется.

Если преобразовывать изображения к разрешению, например, 256x256 или 512x512, то такие дефекты уже отсутствуют (см. рис. 15 и рис. 16):

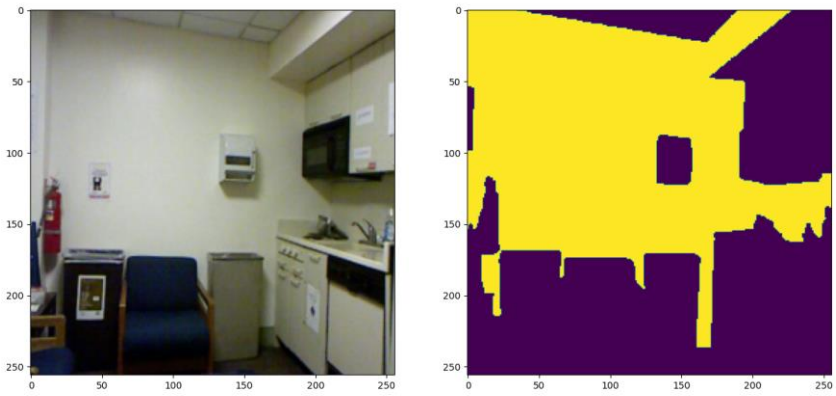

Рис. 15 - Пример фотографии и маски, преобразованных к размерности 256x256

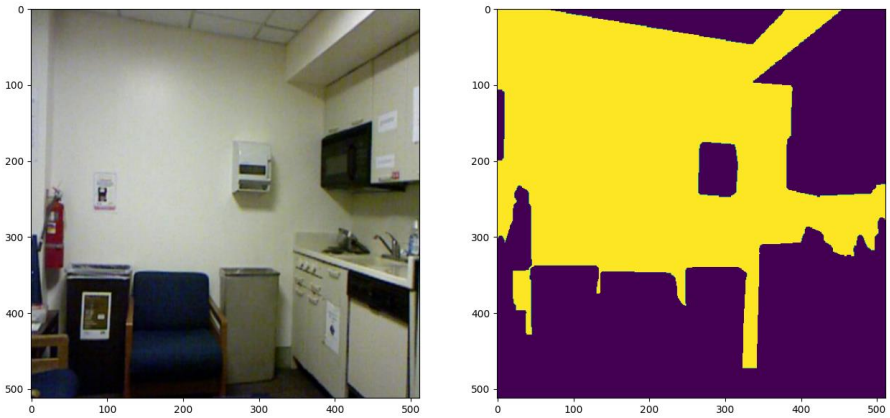

Рис. 16 - Пример фотографии и маски, преобразованных к размерности 512x512

По этой причине далее было проведено обучение модели с теми же параметрами, но с размерностью входного слоя 512x512, графики обучения представлены на рис. 17:

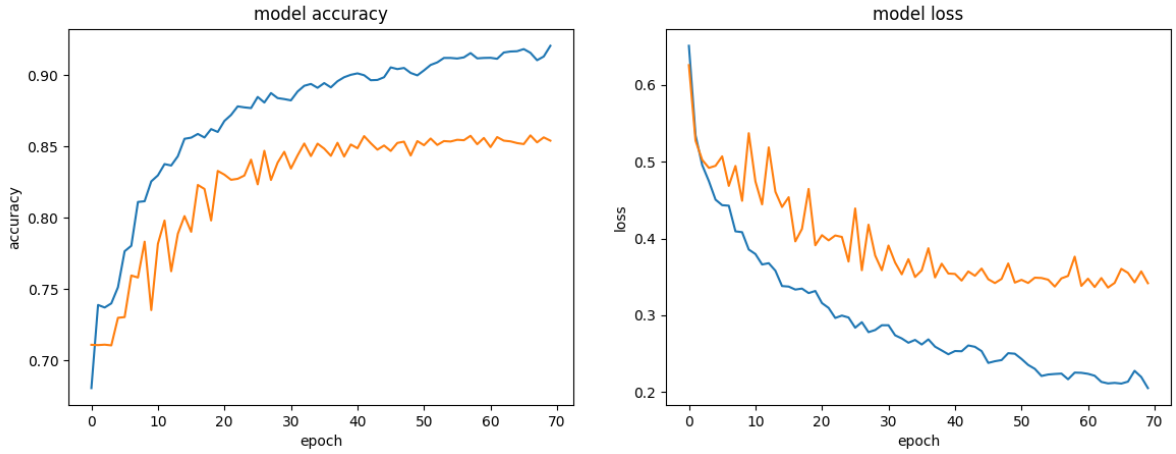

Рис. 17 – Графики точности и потерь при обучении 5-й модели

Результат работы 5-й модели показан на рис. 18 (подробно архитектура модели 6 представлена в табл. Б.2 приложения Б):

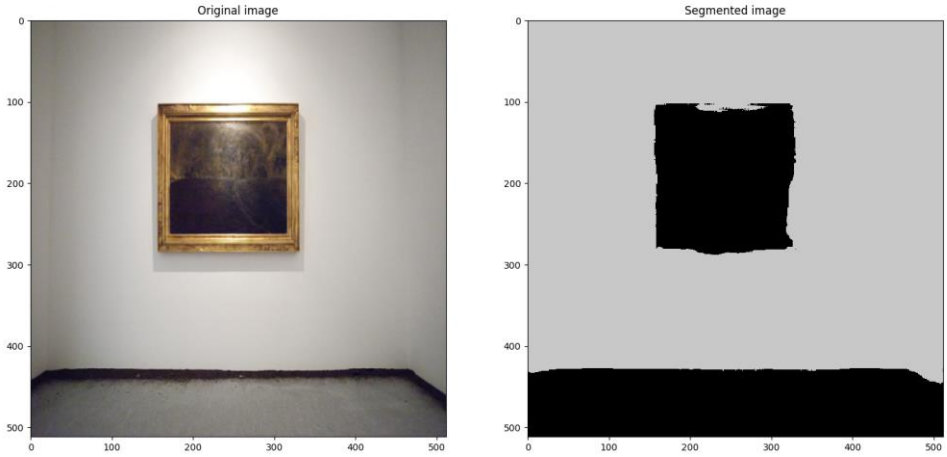

Рис. 18 - Результат работы 5-й обученной модели

Для улучшения точности распознавания стен в архитектуру нейронной сети были добавлены несколько дополнительных слоев свертки и обратной (транспонированной) свертки, после чего была обучена 6-я модель (подробно архитектура модели 6 представлена в табл. Б.3 приложения Б). Графики обучения новой модели (см. рис. 19):

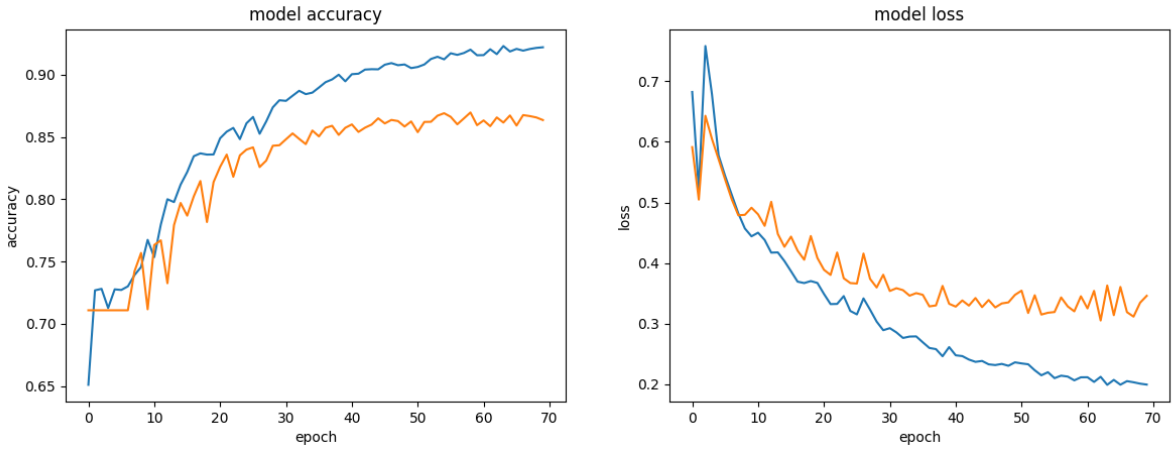

Рис. 19 - Графики точности и потерь при обучении 6-й модели

На рис. 20 и рис. 21 представлены маски одной и той же фотографии, полученные с помощью моделей 5 и 6 соответственно (т.е. до и после обучения с дополнительными слоями):

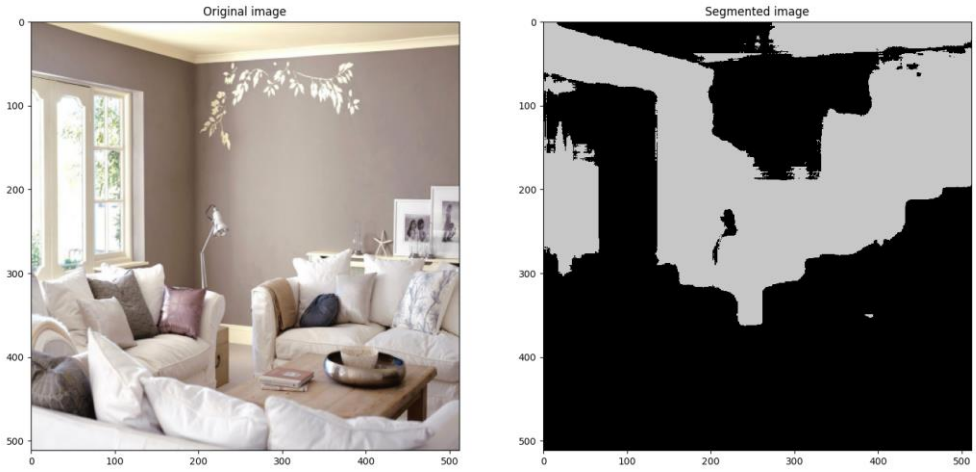

Рис. 20 – Результаты работы на сложном примере до увеличения количества слоев (результат работы 5-й модели)

Рис. 21 – Результаты работы на сложном примере после увеличения количества слоев (результат работы 6-й модели)

Несмотря на выделенную часть потолка и небольшие недостатки на границах, видно, что модель с большим числом слоев, в целом, лучше отображает маску.

Параметры обучения моделей 1-6 представлены в табл. А.1 приложения А.

Глава 4. Решение задачи перекраски

Второй частью решения поставленной задачи является перекраска фотографии под маской. Первоначально был придуман и реализован алгоритм, который перекрашивает пикселы исходной фотографии, с учетом их яркости, формула вычисления каждого из которых по каждой цветовой компоненте:

$$CC_k = color_k \times \frac{max\ pixel_k}{255}, \quad (2)$$

где CC (current color) – вектор, состоящий из трех цветовых компонент рассматриваемого пиксела создаваемого изображения,

$pixel$ – вектор, состоящий из трех цветовых компонент рассматриваемого пиксела исходного изображения,

$color$ – вектор, состоящий из трех цветовых компонент цвета, в который перекрашивается изображение, указывается пользователем,

$k = \overline{1,3}$

Результат работы этого алгоритма представлен на рис. 22:

Рис. 22 – Пример работы программы перекраски

На этих фотографиях видно, что стена сохранила рельеф и тени, а небольшие дефекты на границах стен – изначальная неточность масок датасета, поданных на вход программе.

После этого, были объединены программы сегментации (6-я модель) и перекраски на примере различных изображений, не входящих в датасет (см. рис. 23):

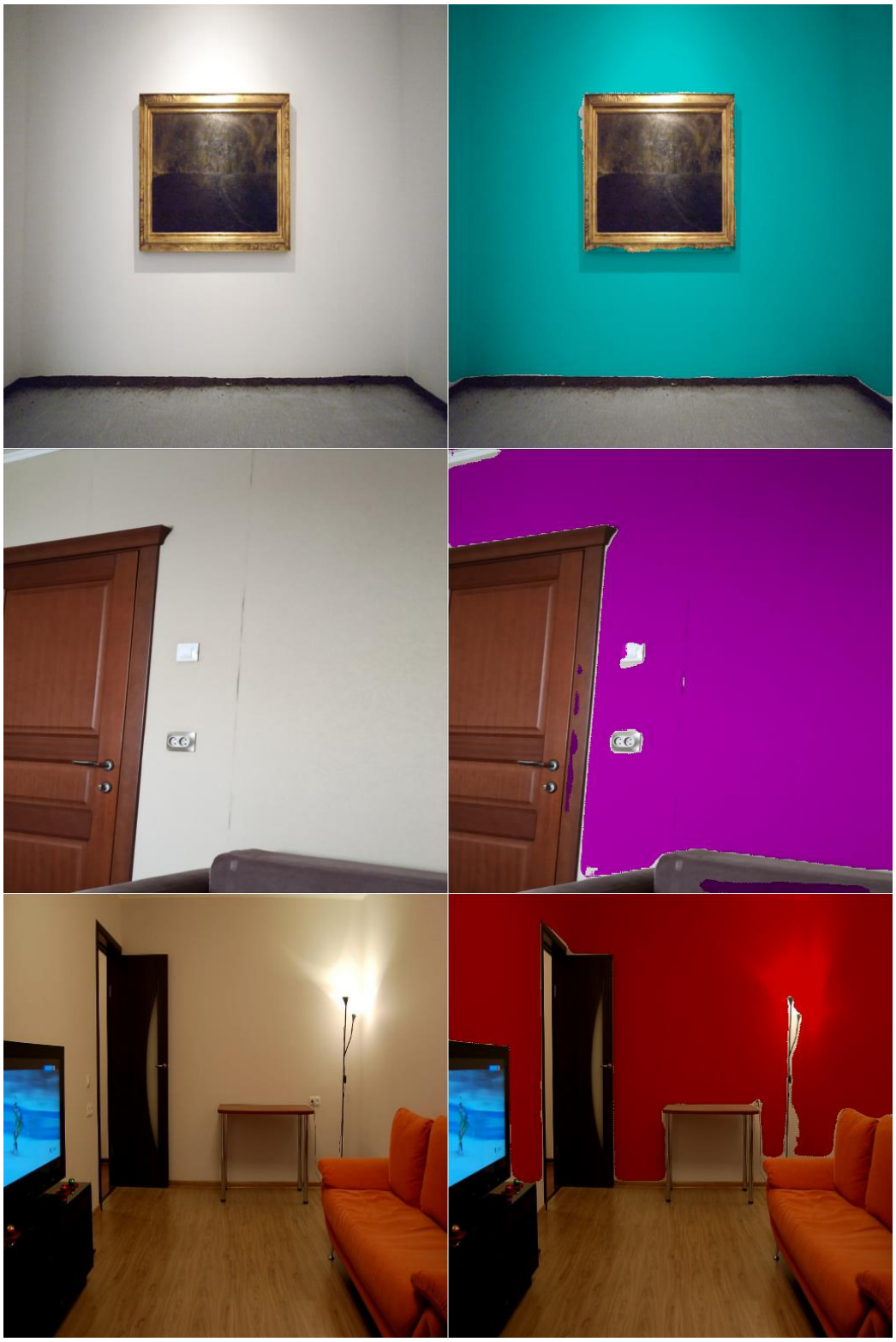

Рис. 23 – Пример работы программы перекраски, объединенной с программой, выполняющей сегментацию (6-я модель)

Для того, чтобы перекрасить в нужный цвет, необходимо в программе поменять значение вектора color, в котором содержится RGB-значение накладываемого цвета.

У такого перекрашивания с учетом рельефа есть один недостаток – накладываемый цвет, который содержится в векторе, является цветом, относительным для каждого пиксела исходного изображения под маской.

Это значит, что, установив значение color = (255, 255, 255), стена не перекрасится в белый цвет, а будет черно-белого оттенка с яркостью, соответствующей исходной. Если выставить, например, color = (255, 0, 0), то пикселы, соответствующие маске, просто потеряют зеленую и синюю составляющую, а красная останется той же, а color = (200, 0, 0) сделает то же самое, только еще немного затемнит.

Т.е. для того, чтобы выставить цвет светлее, чем текущий, необходимо в векторе color выставить некоторые значения больше 255, однако изображение испортится, если значение реальных компонент цвета превысит 255.

Например, если установить color = (400, 400, 400) для достаточно темной фотографии, то она может стать светлее без дефектов (см. рис. 24):

Рис. 24 – Осветление без дефекта превышения максимального значения

Но, если на фотографии стена достаточно светлая, либо на стену падает яркий свет лампы, т.е. цветовые компоненты в некоторых местах стены около максимального значения, то значения некоторых пикселов измененной фотографии будут превышать максимальное, что означает ее неправильное отображение (см. рис. 25):

Рис. 25 – Осветление с дефектом превышения максимального значения

Для того, чтобы избежать этой ошибки, было поставлено максимальное ограничение на цвет пиксела, и результат стал следующим (см. рис. 26):

Рис. 26 – Исправленное осветление

После введения этого ограничения были устранены сбои в цвете из-за превышения максимального значения пиксела, однако, недочет был в том, что вся эта процедура убрала тени под картиной на стене. В связи с этим было предложено изменить принцип перекраски следующим образом: согласно новому разработанному алгоритму, каждая составляющая вектора color варьировалась от 0 до 255, но, при этом, дополнительно был введен параметр ind (indifference), принимающий значение от 0 до 1, который указывает программе, насколько сильно влияет яркость текущего пиксела на новый цвет. Новая формула алгоритма перекраски:

$$CC_k = (1 - ind) \times \left(color_k \times \frac{\max\limits_k pixel_k}{255} \right) + ind \times color_k, \quad (3)$$

где ind (indifference) – коэффициент невосприимчивости к изначальному цвету пиксела.

Можно указать белый цвет, т.е. color = (255, 255, 255) и ind = 0.4 и получить следующий результат (см. рис. 27):

Рис. 27 – Исправленное осветление с регулировкой индифферентности к яркости исходного пиксела

Т.е. параметр ind отвечает за яркость стен. Если нет цели сделать стены светлее, нужно оставить параметр равным нулю, чтобы не потерять видимость теней и рельефа стен. Если же нужно перекрашивать стены в белый или другой светлый цвет, то значение параметра ind нужно ставить, учитывая особенности фотографии: если есть значительный рельеф и тени, стоит указать значение от 0.1 до 0.5, если же нет – до 1.

Выводы

Наилучшим образом в процессе тестирования отработали модели 4 и 6. Модель 4 была обучена на наиболее подходящей из трех выборок изображений датасета, и, в отличие от предыдущих, для ее обучения использовалась аугментация, из-за чего она отработала лучше остальных моделей, которые на выходе выдавали изображение с размерностью 128x128. По сравнению с моделями 1-4, модели 5 и 6 на выходе выдавали изображение более высокого разрешения: 512x512, такой результат ближе к поставленной задаче. При этом, 6-я модель отработала значительно лучше, чем 5-я, из-за увеличенного количества слоев в архитектуре нейронной сети.

Исходя из результатов визуального просмотра ограниченного набора изображений, сформированных программой, выполняющей сегментацию (6-я модель) и перекраску:

- Семантическая сегментация выполняется с точностью около 86%;
- Перекраска изображения происходит корректно, реалистично, с регулировкой яркости;

Возможные улучшения:

- Если найти или собрать более подходящий для задачи датасет, можно повысить точность сегментации;
- Чтобы избежать ситуации недостатка нейронов в сети, стоит расширить ее архитектуру, добавив еще некоторое количество слоев;
- Можно улучшить поведение нейронной сети при анализе граничных точек путем сопоставления каждого пиксела фотографии правильному классу маски;
- Также для максимально реалистичной сегментации, возможно, стоит использовать генеративно-состязательную архитектуру нейронной сети;
- Если разрешение фотографий из датасета примерно 512x512 (как в данной работе), то, для проверки правильности подобранных параметров, сначала можно обучать на этих фотографиях, преобразованных к разрешению 256x256, т.к. времени по сравнению с изначальным разрешением уходит значительно меньше, а качество не теряется на большинстве фотографий, как это происходит с разрешением 128x128;
- Есть смысл обучить модели на нескольких различных входных размерностях фотографий, соответствующих горизонтальным и вертикальным изображениям, для того чтобы измененная фотография не была сильно деформирована.

Заключение

В данной выпускной квалификационной работе был подобран датасет с разнообразными фотографиями комнат и соответствующими им изображениями-масками, на его основе было проведено формирование обучающего и валидационного наборов изображений, которые были наиболее подходящими для задачи.

Затем была построена и реализована нейронная сеть для сегментации, путем различных изменений и модификаций архитектуры нейронной сети было получено 6 различных обученных моделей, наиболее эффективная из которых была выбрана в качестве решения первой части поставленной задачи.

После реализации задачи сегментации, первоначально был разработан и программно реализован алгоритм перекраски изображений, а затем, объединен в один скрипт с сегментирующей программой. После выявления недочетов алгоритма программы перекраски, была разработана и программно реализована его модификация, с помощью которой были устранены ошибки предыдущего варианта программы и добавлен некоторый функционал в виде регулировки яркости.

Несмотря на некоторые неточности работы программы на данном этапе (могут быть слегка закрашены части объектов комнаты или небольшие пробелы на стене, а также неточности на границах), пользователь, в целом, получает преставление о том, как будет выглядеть его комната в том или ином цвете.

Список использованных источников

1. Небольшое исследование свойств простой U-net, классической сверточной сети для сегментации. [Электронный ресурс]. URL: https://www.pvsm.ru/obrabotka-izobrazhenij/302465 (дата обращения: 24.04.2021).

2. Тропченко А.Ю., Тропченко А.А. Методы вторичной обработки и распознавания изображений. Учебное пособие [Книга]. - Санкт-Петербург: Университет ИТМО, 2015. - 215 с.

3. A. Gulli, S. Pal. Deep Learning with Keras [Книга]. – Birmingham – Mumbai: Packt Publishing, 2017. – 318с.

4. Dropout — метод решения проблемы переобучения в нейронных сетях. [Электронный ресурс]. URL: https://habr.com/ru/company/wunderfund/blog/330814/ (дата обращения: 07.03.2021).

5. Landscape – mel chin. [Электронный ресурс]. URL: https://melchin.org/oeuvre/landscape/ (дата обращения: 24.04.2021).

6. Matplotlib: Python plotting – Matplotlib 3.4.2 documentation. [Электронный ресурс]. URL: https://matplotlib.org/ (дата обращения: 23.04.2021).

7. RMRC Reconstruction Meets Recognition Challenge 2014. [Электронный ресурс]. URL: https://cs.nyu.edu/~silberman/rmrc2014/indoor.php (дата обращения: 15.03.2021).

8. scikit-image: Image processing in Python – scikit-image. [Электронный ресурс]. URL: https://scikit-image.org/ (дата обращения: 24.04.2021).

9. S. Pattanayak. Pro Deep Learning with TensorFlow [Книга]. - Berkeley, CA: Apress, 2017. – 398 с.

10. P. Deitel, H. Deitel. Python for Programmers: with Big Data and Artificial Intelligence Case Studies [Книга]. – Pearson Higher Ed USA, 2019. 640с.

Приложение А

Таблица А.1 – параметры обучения моделей

№	Размер выборки	Функция ошибки	Размер батча	Оптимизатор	Шаг обучения	Число нейронов	Аугментация	Разрешение	Колво эпох
1	1400 (140 тестовых)	“Binary-crossentropy”	20	”Adam”	0.001	1 941105	Нет	128x128	50
2	1000 (100 тестовых)		25		0.001	1941105	Нет	128x128	70
3	640 (64 тестовых)		32		0.005	1941105	Нет	128x128	70
4	640 (64 тестовых)		32		0.003	1941105	Да	128x128	70
5	640 (64 тестовых)		32		0.003	1941105	Да	512x512	70
6	640 (64 тестовых)		32		0.003	31110001	Да	512x512	70

Приложение Б

Таблица Б.1 – структура нейронной сети моделей 1-4

Слой	Размерность выхода	Количество параметров	Передается на вход слою
input_1 (InputLayer)	[(None, 128, 128, 3)	0	-
lambda (Lambda)	(None, 128, 128, 3)	0	input_1[0][0]
conv2d (Conv2D)	(None, 128, 128, 16)	448	lambda[0][0]
dropout (Dropout)	(None, 128, 128, 16)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 128, 128, 16)	2320	dropout[0][0]
max_pooling2d (MaxPooling2D)	(None, 64, 64, 16)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 64, 64, 32)	4640	max_pooling2d[0][0]
dropout_1 (Dropout)	(None, 64, 64, 32)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 64, 64, 32)	9248	dropout_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 32)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 32, 32, 64)	18496	max_pooling2d_1[0][0]
dropout_2 (Dropout)	(None, 32, 32, 64)	0	conv2d_4[0][0]
conv2d_5 (Conv2D)	(None, 32, 32, 64)	36928	dropout_2[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 64)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 16, 16, 128)	73856	max_pooling2d_2[0][0]
dropout_3 (Dropout)	(None, 16, 16, 128)	0	conv2d_6[0][0]
conv2d_7 (Conv2D)	(None, 16, 16, 128)	147584	dropout_3[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 128)	0	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 8, 8, 256)	295168	max_pooling2d_3[0][0]
dropout_4 (Dropout)	(None, 8, 8, 256)	0	conv2d_8[0][0]
conv2d_9 (Conv2D)	(None, 8, 8, 256)	590080	dropout_4[0][0]
conv2d_transpose (Conv2DTranspose)	(None, 16, 16, 128)	131200	conv2d_9[0][0]
concatenate (Concatenate)	(None, 16, 16, 256)	0	conv2d_transpose[0][0] conv2d_7[0][0]
conv2d_10 (Conv2D)	(None, 16, 16, 128)	295040	concatenate[0][0]
dropout_5 (Dropout)	(None, 16, 16, 128)	0	conv2d_10[0][0]
conv2d_11 (Conv2D)	(None, 16, 16, 128)	147584	dropout_5[0][0]
conv2d_transpose_1 (Conv2DTranspose)	(None, 32, 32, 64)	32832	conv2d_11[0][0]
concatenate_1 (Concatenate)	(None, 32, 32, 128)	0	conv2d_transpose_1[0][0] conv2d_5[0][0]
conv2d_12 (Conv2D)	(None, 32, 32, 64)	73792	concatenate_1[0][0]
dropout_6 (Dropout)	(None, 32, 32, 64)	0	conv2d_12[0][0]

Продолжение табл. Б.1.

conv2d_13 (Conv2D)	(None, 32, 32, 64)	36928	dropout_6[0][0]
conv2d_transpose_2 (Conv2DTranspose)	(None, 64, 64, 32)	8224	conv2d_13[0][0]
concatenate_2 (Concatenate)	(None, 64, 64, 32)	0	conv2d_transpose_2[0][0] conv2d_3[0][0]
conv2d_14 (Conv2D)	(None, 64, 64, 32)	18464	concatenate_2[0][0]
dropout_7 (Dropout)	(None, 64, 64, 32)	0	conv2d_14[0][0]
conv2d_15 (Conv2D)	(None, 64, 64, 32)	9248	dropout_7[0][0]
conv2d_transpose_3 (Conv2DTranspose)	(None, 128, 128, 16)	2064	conv2d_15[0][0]
concatenate_3 (Concatenate)	(None, 128, 128, 32)	0	conv2d_transpose_3[0][0]
conv2d_16 (Conv2D)	(None, 128, 128, 16)	4624	concatenate_3[0][0]
dropout_8 (Dropout)	(None, 128, 128, 16)	0	conv2d_16[0][0]
conv2d_17 (Conv2D)	(None, 128, 128, 16)	2320	dropout_8[0][0]
conv2d_18 (Conv2D)	(None, 128, 128, 1)	17	conv2d_17[0][0]

Таблица Б.2 – структура нейронной сети модели 5

Слой	Размерность выхода	Количество параметров	Передается на вход слою
input_1 (InputLayer)	(None, 512, 512, 3)	0	-
lambda (Lambda)	(None, 512, 512, 3)	0	input_1[0][0]
conv2d (Conv2D)	(None, 512, 512, 16)	448	lambda[0][0]
dropout (Dropout)	(None, 512, 512, 16)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 512, 512, 16)	2320	dropout[0][0]
max_pooling2d (MaxPooling2D)	(None, 256, 256, 16)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 256, 256, 32)	4640	max_pooling2d[0][0]
dropout_1 (Dropout)	(None, 256, 256, 32)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 256, 256, 32)	9248	dropout_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 256, 256, 32)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 128, 128, 64)	18496	max_pooling2d_1[0][0]
dropout_2 (Dropout)	(None, 128, 128, 64)	0	conv2d_4[0][0]
conv2d_5 (Conv2D)	(None, 128, 128, 64)	36928	dropout_2[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 64, 64, 64)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 64, 64, 128)	73856	max_pooling2d_2[0][0]
dropout_3 (Dropout)	(None, 64, 64, 128)	0	conv2d_6[0][0]
conv2d_7 (Conv2D)	(None, 64, 64, 128)	147584	dropout_3[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 32, 32, 128)	0	conv2d_7[0][0]

Продолжение табл. Б.2.

conv2d_8 (Conv2D)	(None, 32, 32, 256)	295168	max_pooling2d_3[0][0]
dropout_4 (Dropout)	(None, 32, 32, 256)	0	conv2d_8[0][0]
conv2d_9 (Conv2D)	(None, 32, 32, 256)	590080	dropout_4[0][0]
conv2d_transpose (Conv2DTranspose)	(None, 64, 64, 128)	131200	conv2d_9[0][0]
concatenate (Concatenate)	(None, 64, 64, 256)	0	conv2d_transpose[0][0] conv2d_7[0][0]
conv2d_10 (Conv2D)	(None, 64, 64, 128)	295040	concatenate[0][0]
dropout_5 (Dropout)	(None, 64, 64, 128)	0	conv2d_10[0][0]
conv2d_11 (Conv2D)	(None, 64, 64, 128)	147584	dropout_5[0][0]
conv2d_transpose_1 (Conv2DTranspose)	(None, 128, 128, 64)	32832	conv2d_11[0][0]
concatenate_1 (Concatenate)	(None, 128, 128, 128)	0	conv2d_transpose_1[0][0] conv2d_5[0][0]
conv2d_12 (Conv2D)	(None, 128, 128, 64)	73792	concatenate_1[0][0]
dropout_6 (Dropout)	(None, 128, 128, 64)	0	conv2d_12[0][0]
conv2d_13 (Conv2D)	(None, 128, 128, 64)	36928	dropout_6[0][0]
conv2d_transpose_2 (Conv2DTranspose)	(None, 256, 256, 32)	8224	conv2d_13[0][0]
concatenate_2 (Concatenate)	(None, 256, 256, 64)	0	conv2d_transpose_2[0][0] conv2d_3[0][0]
conv2d_14 (Conv2D)	(None, 256, 256, 32)	18464	concatenate_2[0][0]
dropout_7 (Dropout)	(None, 256, 256, 32)	0	conv2d_14[0][0]
conv2d_15 (Conv2D)	(None, 256, 256, 32)	9248	dropout_7[0][0]
conv2d_transpose_3 (Conv2DTranspose)	(None, 512, 512, 16)	2064	conv2d_15[0][0]
concatenate_3 (Concatenate)	(None, 512, 512, 32)	0	conv2d_transpose_3[0][0] conv2d_1[0][0]
conv2d_16 (Conv2D)	(None, 512, 512, 16)	4624	concatenate_3[0][0]
dropout_8 (Dropout)	(None, 512, 512, 16)	0	conv2d_16[0][0]
conv2d_17 (Conv2D)	(None, 512, 512, 16)	2320	dropout_8[0][0]
conv2d_18 (Conv2D)	(None, 512, 512, 1)	17	conv2d_17[0][0]

Таблица Б.3 – структура нейронной сети модели 6

Слой	Размерность выхода	Количество параметров	Передается на вход слою
input_1 (InputLayer)	(None, 512, 512, 3)	0	-
lambda (Lambda)	(None, 512, 512, 3)	0	input_1[0][0]
conv2d (Conv2D)	(None, 512, 512, 16)	448	lambda[0][0]
dropout (Dropout)	(None, 512, 512, 16)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 512, 512, 16)	2320	dropout[0][0]

Продолжение табл. Б.3.

max_pooling2d (MaxPooling2D)	(None, 256, 256, 16)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 256, 256, 32)	4640	max_pooling2d[0][0]
dropout_1 (Dropout)	(None, 256, 256, 32)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 256, 256, 32)	9248	dropout_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 128, 128, 32)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 128, 128, 64)	18496	max_pooling2d_1[0][0]
dropout_2 (Dropout)	(None, 128, 128, 64)	0	conv2d_4[0][0]
conv2d_5 (Conv2D)	(None, 128, 128, 64)	36928	dropout_2[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 64, 64, 64)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 64, 64, 128)	73856	max_pooling2d_2[0][0]
dropout_3 (Dropout)	(None, 64, 64, 128)	0	conv2d_6[0][0]
conv2d_7 (Conv2D)	(None, 64, 64, 128)	147584	dropout_3[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 32, 32, 128)	0	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 32, 32, 256)	295168	max_pooling2d_3[0][0]
dropout_4 (Dropout)	(None, 32, 32, 256)	0	conv2d_8[0][0]
conv2d_9 (Conv2D)	(None, 32, 32, 256)	590080	dropout_4[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 16, 16, 256)	0	conv2d_9[0][0]
conv2d_10 (Conv2D)	(None, 16, 16, 512)	1180160	max_pooling2d_4[0][0]
dropout_5 (Dropout)	(None, 16, 16, 512)	0	conv2d_10[0][0]
conv2d_11 (Conv2D)	(None, 16, 16, 512)	2359808	dropout_5[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 8, 8, 512)	0	conv2d_11[0][0]
conv2d_12 (Conv2D)	(None, 8, 8, 1024)	4719616	max_pooling2d_5[0][0]
dropout_6 (Dropout)	(None, 8, 8, 1024)	0	conv2d_12[0][0]
conv2d_13 (Conv2D)	(None, 8, 8, 1024)	9438208	dropout_6[0][0]
conv2d_transpose (Conv2DTranspose)	(None, 16, 16, 512)	2097664	conv2d_13[0][0]
concatenate (Concatenate)	(None, 16, 16, 1024)	0	conv2d_transpose[0][0] conv2d_11[0][0]
conv2d_14 (Conv2D)	(None, 16, 16, 512)	4719104	concatenate[0][0]
dropout_7 (Dropout)	(None, 16, 16, 512)	0	conv2d_14[0][0]
conv2d_15 (Conv2D)	(None, 16, 16, 512)	2359808	dropout_7[0][0]
conv2d_transpose_1 (Conv2DTranspose)	(None, 32, 32, 256)	524544	conv2d_15[0][0]
concatenate_1 (Concatenate)	(None, 32, 32, 256)	0	conv2d_transpose_1[0][0] conv2d_9[0][0]

Продолжение табл. Б.3.

conv2d_16 (Conv2D)	(None, 32, 32, 256)	1179904	concatenate_1[0][0]
dropout_8 (Dropout)	(None, 32, 32, 256)	0	conv2d_16[0][0]
conv2d_17 (Conv2D)	(None, 32, 32, 256)	590080	dropout_8[0][0]
conv2d_transpose_2 (Conv2DTranspose)	(None, 64, 64, 128)	131200	conv2d_17[0][0]
concatenate_2 (Concatenate)	(None, 64, 64, 256)	0	conv2d_transpose_2[0][0] conv2d_7[0][0]
conv2d_18 (Conv2D)	(None, 64, 64, 128)	295040	concatenate_2[0][0]
dropout_9 (Dropout)	(None, 64, 64, 128)	0	conv2d_18[0][0]
conv2d_19 (Conv2D)	(None, 64, 64, 128)	147584	dropout_9[0][0]
conv2d_transpose_3 (Conv2DTranspose)	(None, 128, 128, 64)	32832	conv2d_19[0][0]
concatenate_3 (Concatenate)	(None, 128, 128, 128)	0	conv2d_transpose_3[0][0] conv2d_5[0][0]
conv2d_20 (Conv2D)	(None, 128, 128, 64)	73792	concatenate_3[0][0]
dropout_10 (Dropout)	(None, 128, 128, 64)	0	conv2d_20[0][0]
conv2d_21 (Conv2D)	(None, 128, 128, 64)	36928	dropout_10[0][0]
conv2d_transpose_4 (Conv2DTranspose)	(None, 256, 256, 32)	8224	conv2d_21[0][0]
concatenate_4 (Concatenate)	(None, 256, 256, 64)	0	conv2d_transpose_4[0][0] conv2d_3[0][0]
conv2d_22 (Conv2D)	(None, 256, 256, 32)	18464	concatenate_4[0][0]
dropout_11 (Dropout)	(None, 256, 256, 32)	0	conv2d_22[0][0]
conv2d_23 (Conv2D)	(None, 256, 256, 32)	9248	dropout_11[0][0]
conv2d_transpose_5 (Conv2DTranspose)	(None, 512, 512, 16)	2064	conv2d_23[0][0]
concatenate_5 (Concatenate)	(None, 512, 512, 32)	0	conv2d_transpose_5[0][0] conv2d_1[0][0]
conv2d_24 (Conv2D)	(None, 512, 512, 16)	4624	concatenate_5[0][0]
dropout_12 (Dropout)	(None, 512, 512, 16)	0	conv2d_24[0][0]
conv2d_25 (Conv2D)	(None, 512, 512, 16)	2320	dropout_12[0][0]
conv2d_26 (Conv2D)	(None, 512, 512, 1)	17	conv2d_25[0][0]