МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«Южный федеральный университет»

Институт математики, механики
и компьютерных наук имени И. И. Воровича

Кафедра алгебры и дискретной математики

Ивахненко Дмитрий Игоревич

СЕМАНТИЧЕСКИЙ АНАЛИЗ ФОТОГРАФИЙ С ПОМОЩЬЮ ГЛУБОКИХ НЕЙРОННЫХ СЕТЕЙ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по направлению подготовки
02.03.02 — Фундаментальная информатика и информационные технологии

Научный руководитель –
к.ф.-м.н., ст.преп. М. В. Юрушкин

Допущено к защите:
заведующий кафедрой _____Штейнберг Б. Я.

Ростов-на-Дону
2020

Содержание

Введение

В рамках данной работы освещается вопрос семантического анализа изображений путем применения глубоких сверточных нейронных сетей. Под семантическим анализом понимается получение из изображения какой-либо интерпретируемой информации: расположение объектов на сцене, принадлежность объектов к заранее заданным классам, наличие на изображении объектов определённого типа и т.п.. Данная тема будет рассмотрена на примере задачи обнаружения и выделения неба на изображениях. Входными данными задачи являются фотографии, сделанные на камеры мобильных устройств. Специализированный домен изображений был выбран с целью упрощения адаптация потенциального решения к применению в конечных продуктах, таких как пользовательское программное обеспечение для смартфонов. Решением задачи выступают сгенерированные для входных фотографий полутоновые изображения. Такое изображение называется сегментационной маской и для каждого пикселя исходной фотографии выражает его принадлежность к региону неба: белый цвет интерпретируется как положительный результат, черный - как отрицательный. В ходе разработки алгоритма решения задачи была исследована эффективность применения различных подходов к генерации подобного рода масок - сегментации. Сравнения эффективности проходило по индексу Жаккара - в западной литературе также встречается название intersection over union, IoU. Для решения задачи сегментации из наиболее эффективных подходов был составлен стек алгоритмов: применение к входному изображению глубокой сверточной сети с последующей корректировкой методами компьютерного зрения полученной маски. В ходе обучения модели искусственной нейронной сети, ИНС, для предотвращения переобучения и улучшения сходимости применялись техники регуляризации, такие как learning rate decaying и cyclic learning rate. Влияние данных подходов на решение также отражено в результатах работы.

В заключительной части хода разработки была рассмотрена возможность адаптация модели, обученной на данных датасета SkyFinder, к выбранному домену изображений. Данная техника имеет название Domain Adaptation и используется для улучшения качества на практических данных.

1. Постановка и описание задачи

Цифровая обработка изображений являются комплексной темой. В нее входят задачи фильтрации, преобразования цвета, яркости и контрастности, морфологической обработка, распознавания и выделения объектов на сцене [1]. Семантический анализу - подход, целью которого является получение интерпретируемой информации высших порядков об изображении. К подзадачам семантического анализа можно отнести классификацию объектов на сцене, детекцию объектов и сегментацию изображений на семантические регионы. В рамках текущей работы будет проведен разбор подзадачи сегментации.

1.1. Задача сегментации

Для возможности цифровой обработки и анализа будем рассматривать представление изображения как трёхмерного массива чисел, имеющего ширину, количество столбцов, и высоту, количество строк, равными ширине и высоте изображения соответственно. На каждой позиции по ширине и высоте будет находится вектор из трех целых чисел, из промежутка [0, 255], что соответствует RGB модели представления цвета пикселя изображения. В таком случае сегментацией изображения будет являться отображение каждого RGB вектора в некоторое целое число. Это число соответствует идентификатору некоторого класса. При применение такого отображения ко всему

изображению получается двумерный массив, ширина и высота которого соответствуют таковым у исходного изображения. Подобный двумерный массив будет разделять изображение на регионы по обозначенному признаку и называться сегментационной маской. Сегментацию возможно рассматривать как попиксельную классификацию объектов.

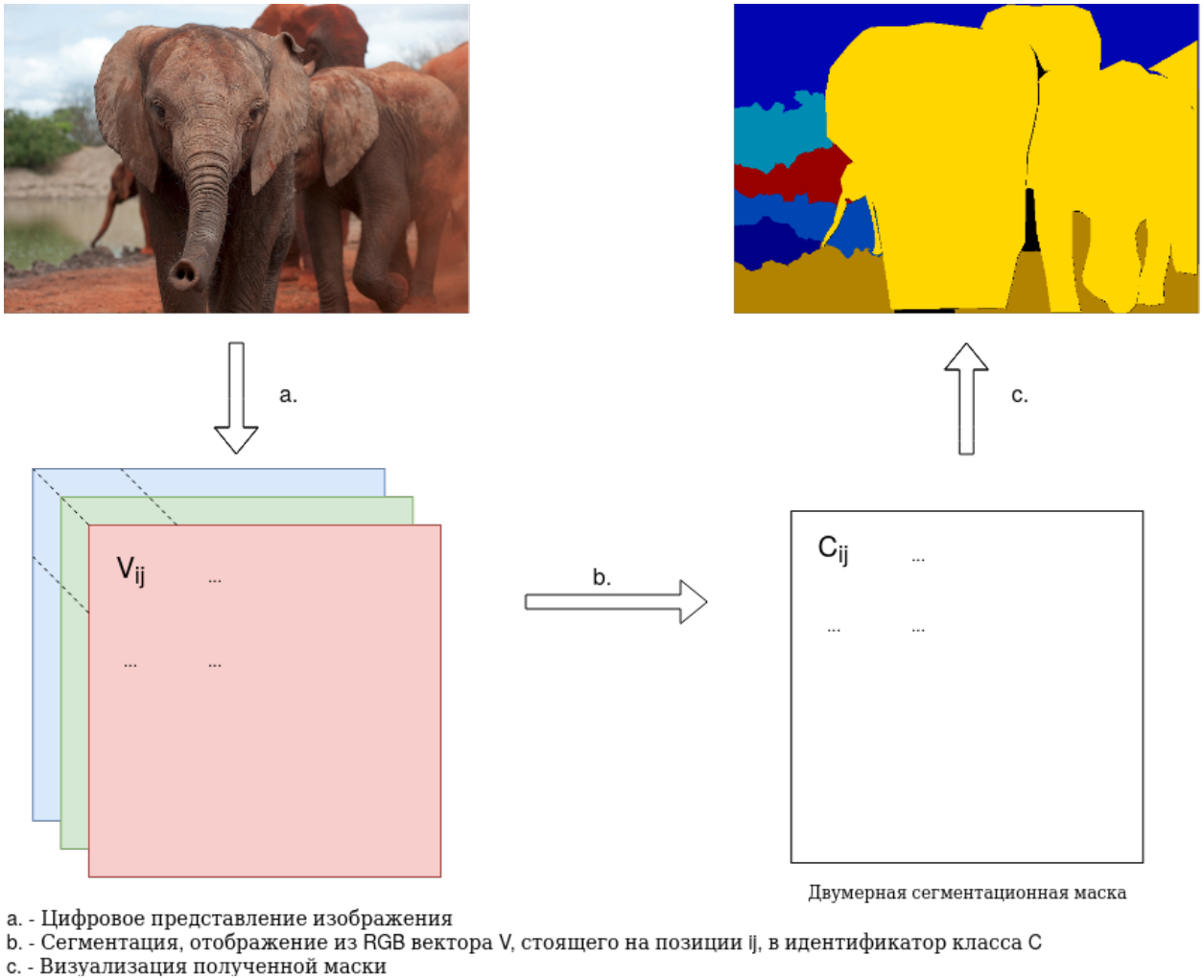


Рисунок 1 — Общий вид сегментации на примере данных датасета COCO

Отметим, что описанное выше отображение может обладать относительно простой природой и учитывать только значение пикселя в конкретной позиции, так и более сложной структурой, исполь-

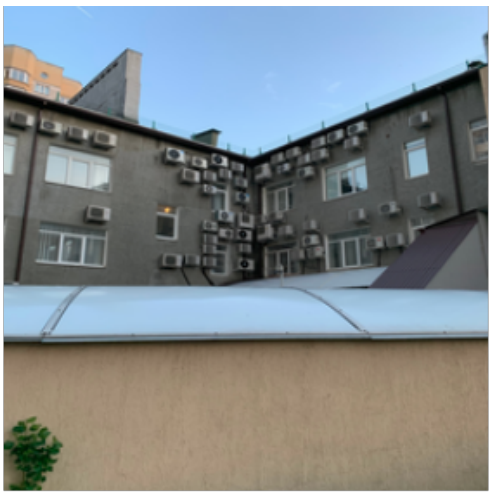
зующей информацию о распределении цветов во всем изображении, положении пикселя на изображении и свойствах соседних пикселей, непосредственно соседствующих с обозреваемым значением или отступающих от него на заданное смещение [2]. На 1 показан общий вид задачи сегментации изображения.

1.2. Выделение неба в рамках задачи сегментации

Описанная в введение задача выделения региона неба на входном изображении может быть рассмотрена как задача сегментации. Так как в данном случае результирующих классов всегда два - класс принадлежности и обратный ему -, то имеется более узкий случай сегментации - бинарная. Итоговая маска будет содержать только значения 0 и 1, представляя собой однобитовое бинарное изображение, что можно считать вырожденным полутоновым.

Таким образом, решение задачи выделения неба сводится к нахождению отображения из вектора цветов для каждого пикселя в целое число из промежутка [0, 1]. Данное отображение возможно получить как алгоритмами машинного зрения, так и с помощью использования моделей глубоких сверточных нейронных сетей. В данной работе приводится решение методом нейронных сетей, при этом алгоритмы компьютерного зрения используются для корректировки полученной маски. Под корректировкой здесь понимается обнаружение и удаление ложноположительных регионов неба небольшой площади.

1. Результат работы сети



2. Исходное изображение

3. Коррекция результата

4. Размеченная маска

Рисунок 2 — Пример, демонстрирующий работу алгоритма

Практическое применение подобного решения можно найти в пользовательских приложениях эстетической обработки пейзажных фотографий для смартфонов, в автоматических системах мониторинга воздушного пространства, при извлечении семантической информации высшего порядка для использования в иных методах анализа и обработки изображений [3].

2. Обзор предметной области

Задача семантического анализа изображений имеет широкое применение при решении различных прикладных и исследовательских проблем [4] [5] [6] [7], в связи с чем активно изучается. Сегментации изображений, в частности, нашли применения в таких областях как проведение хирургических операций, автопилотируемый транспорт, автоматизированное картографирование местности [2]. Ниже приведен краткий обзор исследований, темы которых связаны с поставленной задачей. Результаты этих исследований в разной степени использовались для построения решения.

2.1. Сегментация изображений

До начала активного применения глубоких нейронных сетей в задачах сегментации использовались методы компьютерного зрения, основанные на применении порогов бинаризации для полутоновых изображений, выявлении признаков, кластеризации методом k-средних [8] [9] [10]. Каждый из этих подходов имеет свои преимущества, метод бинарной сегментации полутоновых изображений до сих пор успешно применяется в области анализа медицинских данных [11]. Но применение данных подходов к задаче выделения границы между объектами путем сегментации показало худшие результаты в сравнении с FCN, полностью сверточными глубокими сетями [12].

Современные решения задачи сегментации в различных областях зачастую опираются на применение нейронных сетей [13]. Наиболее распространёнными архитектурами являются Unet, DeepLab, RefineNet [14] [15] [16]. Имеются исследования применения архитектуры RefineNet для определения региона неба на датасете SkyFinder [17].

2.2. Способы регуляризации

Помимо специализированных решений задачи сегментации, были рассмотрены также общие методики, применяемые для обучения глубоких нейронных сетей. При высокой сложности модели, в процессе обучения она может начать отражать в ответах шум в тренировочных данных [18] [19]. Данное явление называется переобучением, overfitting. С целью снизить вероятность его проявление применяются разнообразные техники регуляризации: dropout, нормализация значений между слоями сети, настройка гиперпараметров процесса обучения [20] [21] [22].

3. Метод решения

Конечное решение задачи сегментации имеет комплексное строение. Изображение, поступающее на вход решающему алгоритму, обрабатывается ИНС. Результатом такого применения является двумерный массив пар. Каждое значение в паре обозначает вероятность принадлежности одному из двух классов. Для получения сегментационной маски индекс максимального из двух значений расценивается как идентификатор класса: 1 для региона неба, 0 - обратный ему. В ходе экспериментов было выявлено наличие артефактов в результатах классификации. Для части изображений имелись ложноположительные регионы неба небольшой площади. Для их устранения последовательно применялись алгоритмы FindCounters и DrawCounters из состава библиотеки OpenCV.

Для обучения сети использовался датасет SkyFinder. Для улучшения показателя IoU на целевом домене изображений использовалась техника Unsupervised Domain Adaptation. Это позволило уменьшить количество ложноположительных и ложноотрицательных регионов без дополнительной разметки обучающий выборки из фотографий, сделанных на мобильные устройства.

3.1. Глубокая сверточная сеть

Как было отмечено выше, текущие исследования указывают на преимущества глубоких сверточных сетей в задачах семантического анализа перед классическими алгоритмами компьютерного зрения. Глубокими сверточными сетями называют подмножество искусственных нейронных сетей, все полносвязные слои которых заменены сверточными.

Архитектура сети для решения задачи сегментации представляется двумя частями. Отбор признаков, downsampling path или енкодер, и генерализация результатов отбора, upsampling path или декодер. Для решения проблемы потери информации в сверточных слоях, карты признаков с некоторых уровней нисходящей части объединяют с противоположными входами восходящей части. На 3 приведена общая схема сети для задач сегментации.

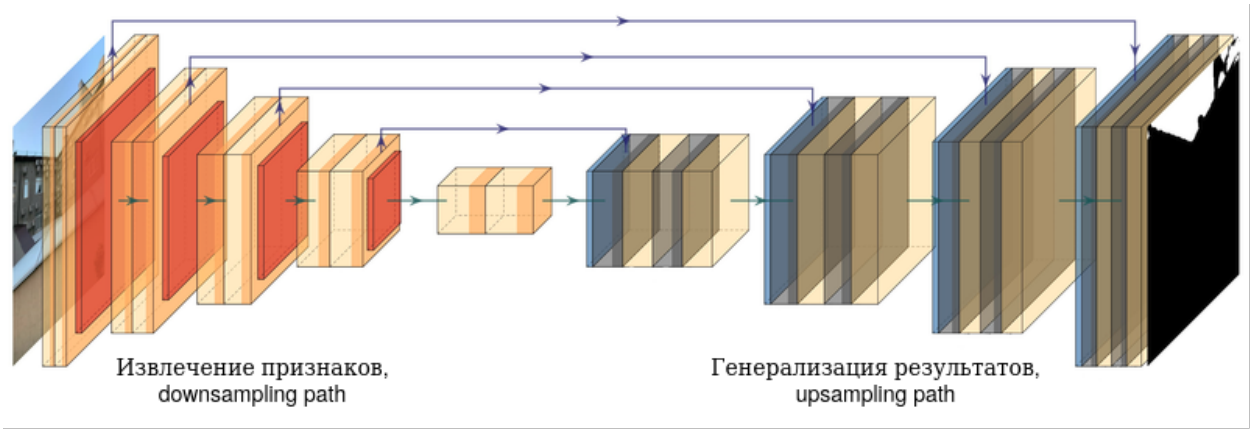


Рисунок 3 — Общий вид глубокой сверточной сети для задачи сегментации

В нисходящей части используются сверточные слои (отмечены желтым цветом на 3), слои с выбором наибольшего значения по ядру - max pooling (отмечены красным цветом на 3). На сверточных слоях к входным данным применяются ядра свертки. Количество ядер определяет для слоя количество выходных карт признаков. Выходные кар-

ты признаков подаются на вход следующему слою. К выходным картам применяется функция активации (отмечены оранжевым цветом на 3). Значения в ядрах свертки являются обучаемыми параметрами сети. Ниже приведена формула для простого случая двумерного входного массива и двумерного ядра свертки (1) при единичном шаге.

$$G_{m,n} = (f * h)[m, n] = \sum_j \sum_k h_{j,k} f_{m+j,n+k} \tag{1}$$

где f - входной массив данных, h - ядро свертки, G - выходная карта признаков.

Размеры каждой карты определяются размерами входных данных, размерами ядра, размером добавочного отступа по краям массива данных и сдвигом, на которое ядро смещается по массиву (2).

$$output = \frac{input + 2 * p - k}{s} + 1 \tag{2}$$

где output - размеры выходной карты признаков, input - размеры входного массива данных, p - величина отступа, добавляемого к краям массива (padding), k - размеры ядра свертки, s - сдвиг ядра.

В слоях с выбором наибольшего значения ядро определяет только размеры окна, в котором выбирается максимальный элемент (3). Операции сложения и умножения на этом слое не используются.

$$G_{i,j} = \max_{(k,l) \in N} F_{i+k,j+l} \tag{3}$$

где

$$N = \{0, .., \text{kernel size}\} \tag{4}$$

G - выходная карта признаков, F - входной массив данных.

В качестве функции активации используется rectified linear unit, ReLU (5).

$$g(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{5}$$

В upsampling части сети используются, помимо сверточных, транспонированные сверточные слои (отмечены синим цветом на 3). Прежде чем описать оператор транспонированной свертки, необходимо рассмотреть применение сверточного ядра к входным данным, как произведение матриц. Рассмотрим на примере ядра свертки размером 3×3 и входного массива размером 4×4. Значение сдвига равно 1, дополнение отступом не используется.

$$\begin{pmatrix} K_{0,0} & K_{0,1} & K_{0,2} & 0 & K_{1,0} & K_{1,1} & K_{1,2} & 0 & K_{2,0} & K_{2,1} & K_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & K_{0,0} & K_{0,1} & K_{0,2} & 0 & K_{1,0} & K_{1,1} & K_{1,2} & 0 & K_{2,0} & K_{2,1} & K_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & K_{0,0} & K_{0,1} & K_{0,2} & 0 & K_{1,0} & K_{1,1} & K_{1,2} & 0 & K_{2,0} & K_{2,1} & K_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & K_{0,0} & K_{0,1} & K_{0,2} & 0 & K_{1,0} & K_{1,1} & K_{1,2} & 0 & K_{2,0} & K_{2,1} & K_{2,2} \end{pmatrix}$$

(6)

$$\begin{pmatrix} W_{0,0} & W_{0,1} & W_{0,2} & W_{0,3} & W_{1,0} & W_{1,1} & W_{1,2} & W_{1,3} & W_{2,0} & W_{2,1} & W_{2,2} & W_{2,3} & W_{3,0} & W_{3,1} & W_{3,2} & W_{3,3} \end{pmatrix}$$

(7)

В (6) введена матрица K, ненулевые элементы которой представлены значениями из ядра свертки. $K_{i,j}$ - элемент, стоящий в ядре на позиции i, j. В (7) введена матрица I, полученная путем перекомпоновки входного массива данных в вектор по строкам. Тогда карту при-

знаков G размером 2×2 можно получить путем умножения матрицы K на I^T.

$$K * I^T = \begin{pmatrix} \sum_j \sum_k h_{j,k} f_{j,k} \\ \sum_j \sum_k h_{j,k} f_{j,1+k} \\ \sum_j \sum_k h_{j,k} f_{1+j,k} \\ \sum_j \sum_k h_{j,k} f_{1+j,1+k} \end{pmatrix} \tag{8}$$

Чтобы из (8) получить (1) необходимо записать матричное произведение, избавиться от нулевых слагаемых и объединить оставшиеся под знаками сумм.

$$K * I^T = \begin{pmatrix} G_{0,0} \\ G_{0,1} \\ G_{1,0} \\ G_{1,1} \end{pmatrix} \tag{9}$$

Введем транспонированную свертку для входных данных F размером 2×2, ядра свертки H 3×3 и выходной карты признаков G 4×4,

как произведение K^T, полученной из H аналогично (6), на F_0^T, полученный из F аналогично (7).

$$K^T * F_0^T = \begin{pmatrix} G_{0,0} \\ G_{0,1} \\ G_{0,2} \\ G_{0,3} \\ G_{1,0} \\ G_{1,1} \\ G_{1,2} \\ G_{1,3} \\ G_{2,0} \\ G_{2,1} \\ G_{2,2} \\ G_{2,3} \\ G_{3,0} \\ G_{3,1} \\ G_{3,2} \\ G_{3,3} \end{pmatrix} \tag{10}$$

Таким образом (10) позволяет восстанавливать исходное количество пикселей для их классификации по выделенным признакам [23]. Отметим здесь, что сверточные слои в восходящей части не уменьшают размеры данных по высоте и ширине за счёт использования паддинга.

После выходного слоя размеры массива данных равняются $N \times M \times C$, где N, M - размеры исходного изображения, а C - количество классов. К полученным данным применяется логистическая функция, обобщённая на многомерный случай, softmax (11). Вычисления осуществляются для каждой позиции $[n, m]$ по размерности C.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{11}$$

В результате каждый вектор $\bar{v}$ из массива данных по размерности C обладает следующими свойствами:

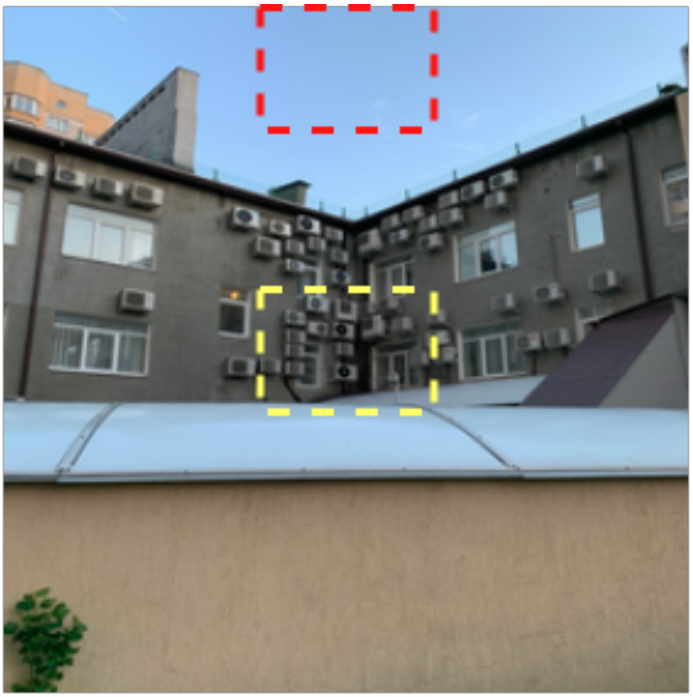
- $v_i \in [0, 1] \forall i \in [0, C]$

- $\sum_{i=0}^{C} v_i = 1$

Ниже приведен (4) отладочный вывод данных до и после softmax для программы, реализующей обучения глубокой сверточной сети с ResNet34 блоком в качестве downsampling части и LinkNet в качестве upsampling.

Красным отмечен регион, из которого случайным образом брался пример 1.

Желтым - пример 2.



```
Pre-softmax state; Size: torch.Size([2, 256, 256]),
1. tensor([-9.2658939, 10.0882759], dtype=torch.float64),
2. tensor([ 4.1472898, -4.4603243], dtype=torch.float64)

Post-softmax state; Size: torch.Size([2, 256, 256]),
1. tensor([    0.0000000,     1.0000000], dtype=torch.float64),
2. tensor([    0.9998173,     0.0001827], dtype=torch.float64)
```

Рисунок 4 — Отладочная информация с визуализацией

3.2. Регуляризация на уровне архитектуры

Как было отмечено в части обзора решений, при обучении сети может возникнуть overfitting. Состояние модели, в котором она демонстрирует значительно более высокие метрики качества на тренировочных данных, чем на тестовых. Причиной такого поведения может быть запоминание шумов данных. К ним могут быть отнесены статистические выбросы, неравномерное представление классов в выборке, фактическая природа данных. В частности, при преобладании определённого признака в данных вероятно проявление ковариантного сдвига по признаку (Covariate Shift) [24]. Подобное смещение приводит к смещению на уровне внутренних слоев. Для предотвращения влияния данные между слоями приводят к нормализованному состоянию (12). Нормализация совершается по пакету, batch.

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$
2. $\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$
3. $\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$
4. $\mathrm{BN}_{\gamma,\beta}(x_i) = \gamma\hat{x}_i + \beta$ (12)

Здесь $x_i \in X$, где X - текущий пакет входных данных. Этапы 12.1, 12.2, 12.3 применяются только на этапе обучения сети. Параметры γ и β являются обучаемыми. При инференсе сети они обеспечивают соответствие среднего и смещения для входных данных аналогичными показателями тренировочных пакетов. На 5 продемонстрировано сравнение процессов обучения с и без описанной техники.

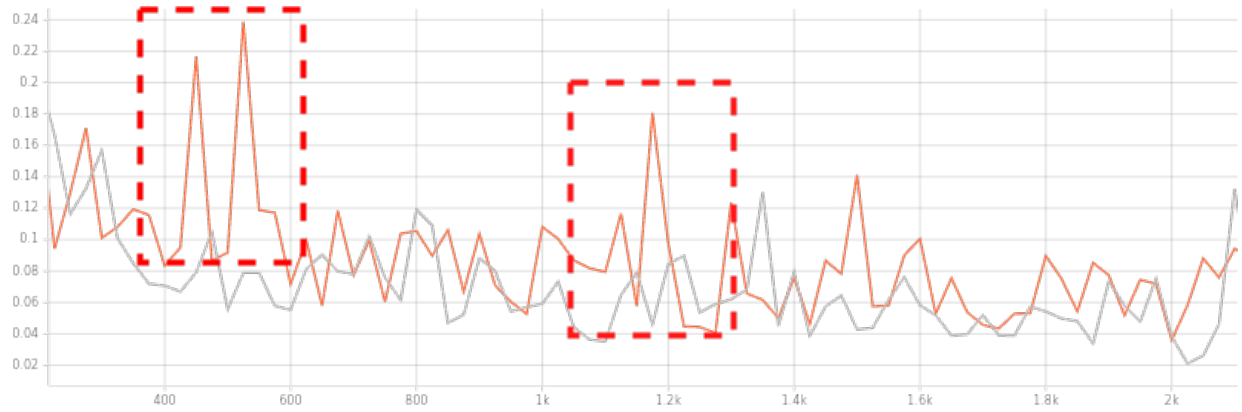


График значения функции потерь к количество пройденных образцов
Графики:
1. Оранжевый - без нормализации по пакету
2. Серый - с нормализацией по пакету

Рисунок 5 — Улучшение сходимости нормализацией

3.3. Извлечение признаков, енкодер часть

Для выделение признаков рассматривались сети архитектуры ResNet. Отличительной чертой данной архитектуры являются residual block и наличие skip connection. В состав одного блока последовательно входят:

1. Сверточный слой с паддингом
2. Нормализация по пакету
3. Активационная функция
4. Сверточный слой с паддингом
5. Нормализация по пакету

Результат блока объединяется с входными данными. Такое решение позволяет предотвратить затухание градиента при обратном распространении ошибки [25].

Для снижения размеров массива данных в архитектуре ResNet используются блоки, сочетающие в себе сверточный слой с ядром 1×1, без паддинга и сдвигом в 2 и нормализацию по пакету.

В итоговое решение в качестве downsampling path вошла архитектура ResNet34. Она обладает меньшим количеством тренируемых параметров в сравнении с ResNet50, ResNet101, ResNet152. При этом демонстрируется достаточно высокое значение метрики качества.

3.4. Генерализация признаков, декодер часть

В downsampling части сети по извлеченным в енкодере признакам генерируется маска. Для выбора декодера было проведено сравнение двух архитектурных решений: Unet [14] и LinkNet [26]. Оба подхода демонстрируют схожие значения метрики качества: усреднено 95.3 для LinkNet против 95.8 для Unet. При этом за счет меньшего количества тренируемых параметров LinkNet имеет меньшее время обучения (7) и быстрее обрабатывает изображение в инференсе. Unet имеет $26,730,306$ тренируемых параметра, LinkNet - $21,794,850$. Аналогично енкодер части, декодер имеет блочную структуру. Состав каждого блока на примере варианта с использованием LinkNet подхода:

- Сверточный слов с ядром 1×1
- Нормализация по пакету
- Активационная функция
- Транспонированная свертка
- Нормализация по пакету
- Активационная функция
- Сверточный слой с ядром 1×1

- Нормализация по пакету
- Активационная функция

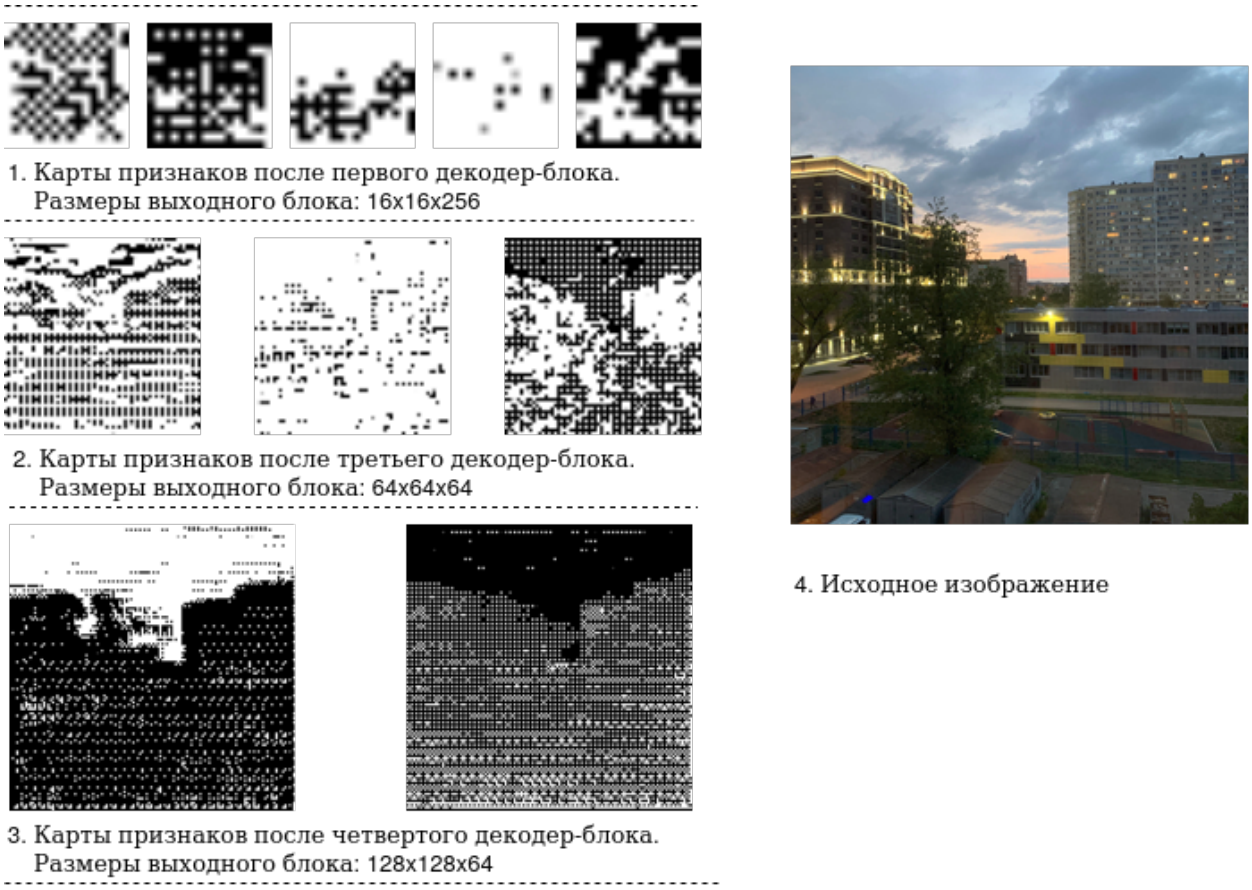


Рисунок 6 — Визуализация выходных карт признаков для различных уровней декодер части

Рисунок 7 — Отладочная информация процесса обучения

3.5. Функция потерь и метрика качества

Для обучения сети по прецедентам методом обратного распространения ошибки необходимо ввести функцию потерь. Данная функция отражает отклонение полученного сетью результата от образца. Минимизации отклонения можно добиться с помощью градиентого спуска.

В данной работе в качестве функции потерь выступает логистическая функция ошибки (13). Встречаются так же названия бинарная кросс-энтропия, логлосс.

$$f(y, \tilde{y}) = \begin{cases} -\log(y) & \tilde{y} = 1 \\ -\log(1-y) & \tilde{y} = 0 \end{cases} \tag{13}$$

В приведённой формуле y - это предсказанный результат, а $\tilde{y}$ - достоверный образец. Значения образца 1 и 0 отражают принадлеж-

ность к целевому классу. При обобщении функции на случай пакетного обучения получаем (14).

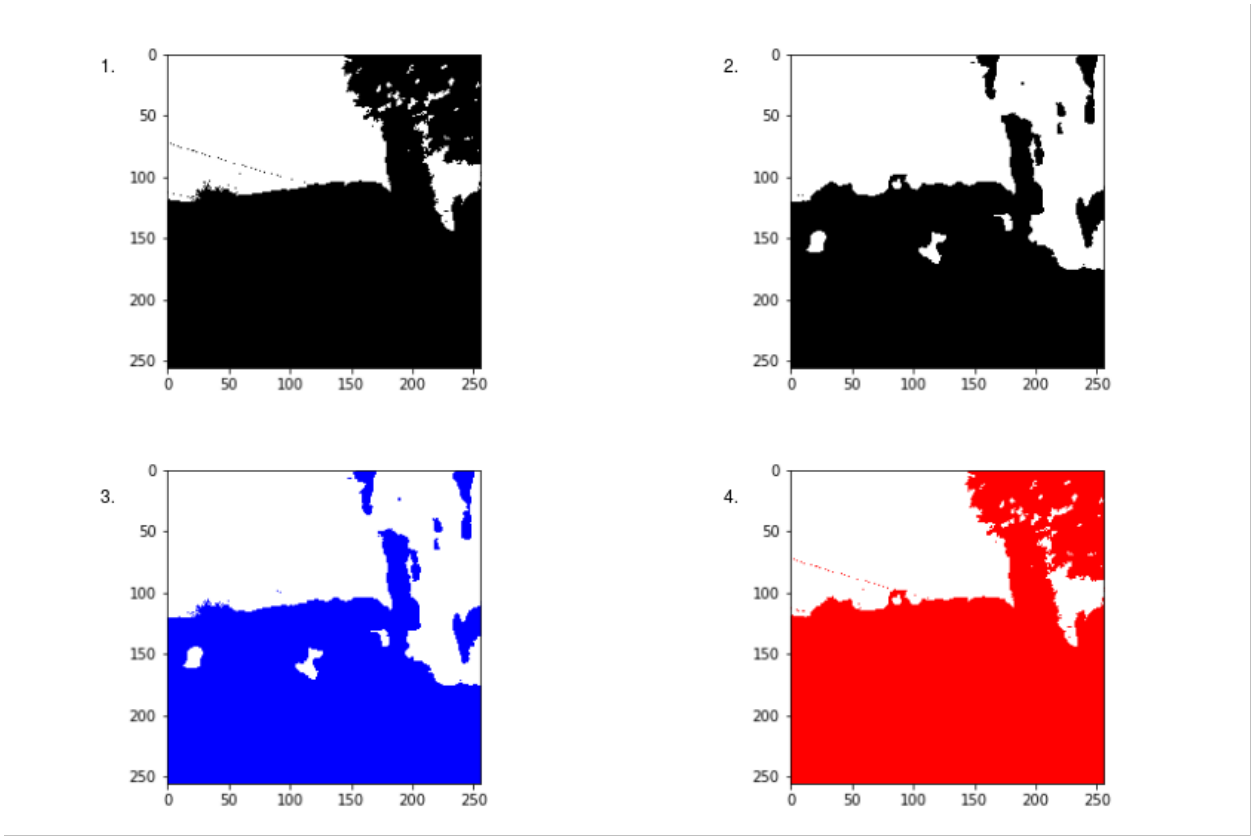
$$f(y, \tilde{y}) = -\frac{1}{n}\sum_{i=0}^{n}(\tilde{y_i} \cdot \log(y_i) + (1 - \tilde{y_i}) \cdot \log(1 - y_i)) \tag{14}$$

Здесь $y, \tilde{y}$ - пакеты размерами n ответов и образцов соответственно. При пакетном обучении обновление весов происходит не после каждого образца, а после каждого пакета - набора образцов.

Для оценки работы сети использовалась метрика intersection over union, индекс Жаккара. Для каждой пары ответ-образец попиксельно вычисляются площади классифицированных сегментов. Значением индекса для класса является отношение площади пересечения к площади объедения сегментов в паре в процентном отношении (15).

Замечу, данная метрика чувствительная не только ложноотрицательным ошибкам при классификации. При ложноотрицательном срабатывании для класса будет уменьшаться индекс за счет уменьшения площади пересечения. При ложноположительном выделении пикселя в класс будет увеличиваться площадь пересечения, что так же негативно скажется на итогом значении метрики для класса.

$$\mathrm{IoU} = \frac{\mathrm{S}_{intersection}}{\mathrm{S}_{union}} \cdot 100\% \tag{15}$$



Визуализация площадей пересечения и объединения.
1. Образец
2. Сгенерированная маска.
   Сеть в предобученном состоянии для наглядной разницы между пересечением и объединением.
3. Пересечение
4. Объединение
Значение метрики для примера: 79.2986

Рисунок 8 — Визуалзиация intersection over union

3.6. Регуляризация на уровне гиперпараметров сети

Для улучшения сходимости и предотвращения явлений предобучения или переобучения можно добавлять различные гиперпараметры к сети такие, как learning rate и momentum.

$$\omega_i = \omega_j - \alpha \cdot \frac{\delta f(y, \tilde{y})}{\delta \omega_j} \tag{16}$$

При оптимизации функции потерь градиентным спуском движение к минимуму определяется градиентом. Learning rate, коэффи-

циент α при градиенте в (16), отражает степень смещения относительно новых значений. При достаточно большом значение ухудшится сходимость, но вероятность остаться в локальном минимуме будет меньше. При маленьких значений коэффициента сходимость к минимуму улучшается, но и возникает проблема остаться в локальном минимуме. В данном решение были рассмотрены два подхода к настройке данного значения в процессе обучения сети: learning rate decaying и cyclic learning rate.

При использовании learning rate decaying изначальное значение learning rate устанавливается относительно высоким. В течении всего обучения это значения поэтапно уменьшается. В качестве функции изменения коэффициента была выбрана fixed stepsize (17), описанная в [27].

$$\alpha_i = \alpha_j^c \tag{17}$$

Переход с шага j на шаг i путем возведения learning rate в степень осуществляется при разнице между предыдущем и текущим значениями функции потерь меньшей ε. В рамках данной работы c равнялось 2, ε равнялось 0.01.

При cyclic learning rate значение постепенно снижается до определённого порога, а затем снова повышается. Подход one cycle policy, рассмотренный в [20], предлагает на протяжении всего процесса обучения расположить один полный цикл увеличения-уменьшения. Для инициализации метода необходимо определить максимальное значение гиперпараметра, уровень относительно максимального значения, с которого learning rate будет увеличиваться. Также по завершении полного цикла необходимо дополнительного уменьшить learning rate для дополнительного снижения отклонения функции потерь. Ниже (9) приведен график изменения параметров learning rate и momentum в ходе one cycle policy.

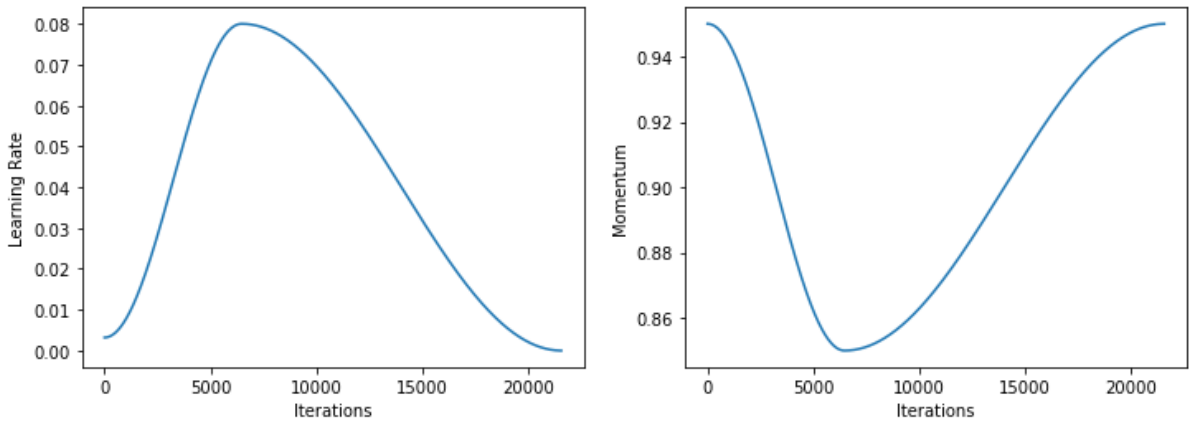


Рисунок 9 — Отладочная информация процесса обучения

Подход применения гиперпараметра momentum заключается в использовании значения обновления весов с предыдущей итерации (18).

$$\omega_i = \omega_j - (\alpha \cdot \frac{\delta f(y, \tilde{y})}{\delta \omega_j} + \gamma \cdot \triangle \omega_k) \tag{18}$$

Здесь $\triangle \omega_k$ - изменение тренируемого параметра ω на предыдущей итерации, γ - коэффициент momentum. Использование данной техники помогает удержать спуск к минимуму при случайных шумах в текущем пакете данных.

3.7. Корректировка маски

Для устранения артефактов сегментации сетью использовались функции FindContours и DrawContours из состава библиотеки OpenCV. В основе FindContours лежит алгоритм следования границы, описанный в [28].

После нахождения контура, для региона, находящегося внутри него, вычисляется площадь. При превышении эмпирически установленного порога, применяется DrawContours. Основная задача такой

доработки устранить влияние отражающих поверхностей на изображении на конечное качество сегментации 10.

Применение FindContours позволяет устранить замкнутые внутренние ложноположительные регионы, причиной которых могло послужить наличие отражающих поверхностей на изображении (отмечено красным пунктиром).
При этом ложноположтительные открытые регионы такой подход исправить не может (оранжевый пунктир).

Рисунок 10 — Пример работы FindContours

4. Данные

Эксперименты с обучением модели глубокой сверточной сети проводились на двух наборах данных. Первый набор - это SkyFinder, открытый датасет, с размеченными под сегментацию неба изображениями. Второй - объединение первого с изображениями, сделанными на камеру смартфона. Примешивание к SkyFinder данных из целевого домена способствовало улучшению метрик качества на исходной валидационной выборке, так как модель при обучении учитывала особенности цветопередачи и отличия в распределении двух классов на новых изображениях. Таким образом увеличение разнообразия в данных уменьшило ковариантный сдвиг по доминирующим признакам в исходном датасете.

4.1. Датасет SkyFinder

SkyFinder представляет собой набор из 90.000 фотографий [29]. Все фотографии сделаны на статичные веб-камеры, расположенные вне зданий. Для каждого изображения имеется размеченная бинарная маска с описанным в постановке задачи свойством: 1 обозначает класс принадлежности, 0 - обратный ему. В верхней части изображений преобладает регион неба. Средний процент пикселей относящихся к классу принадлежности равен 41 со стандартным отклонением в 16 процентов. Изображения покрывают широкий диапазон освещённости и погодных условий, что препятствует переобучению на конкретных состояниях освещённости сцены. При этом из-за статичности камер и их ограниченного количества - всего камер в отборе участвовало 53 штуки - модель при обучении демонстрирует частые ложные срабатывания на близко расположенных детализированных объектах. Ниже приведены примеры изображений из датасета (11).

Рисунок 11 — Примеры изображений из SkyFinder

4.2. Датасет с синтетической разметкой

В качестве дополнительных данных были использованы изображения с открытых интернет ресурсов. Изображения были вручную очищены от тех, которые не имели в себе сегментов неба. Небольшая часть из оставшихся была размечена с помощью графических редакторов. Для разметки остальных использовалась сеть, обученная на данных из SkyFinder. Такой подход позволяет разительно сократить временной ресурс, затрачиваемый на подготовку данных. Подобная техника описывается в [30] на примере миграции между датасетами CamVid и Cityscapes.

Для обучения сети на новом наборе данных был написан отдельный загрузчик на базе загрузчиков библиотеки глубокого обучения PyTorch. Особенностью нового загрузчика является возможность совмещать в рамках одного пакета данные из разных источников на физическом диске. При этом количественное отношение между разными данными представляется к настройке пользователю. В рамках данной работы источников данных было два. На пакет величиной 17 приходилось 13 изображений из датасета SkyFinder и 4 из данных с синтетической разметкой. Термин "синтетическая"обусловлен использованием сети при разметки данных. Соотношение в пакете было подобрано эмпирически. Учитывалось, что большое количество изображений из новых данных может оказать негативный эффект на метриках качества, из-за погрешностей в разметке.

Для получения данных с синтетической разметкой использовалась сеть, оубченная только на SkyFinder. При этом на объединение данных сеть обучалась с нуля, не используя до этого подготовленные веса.

5. Результаты

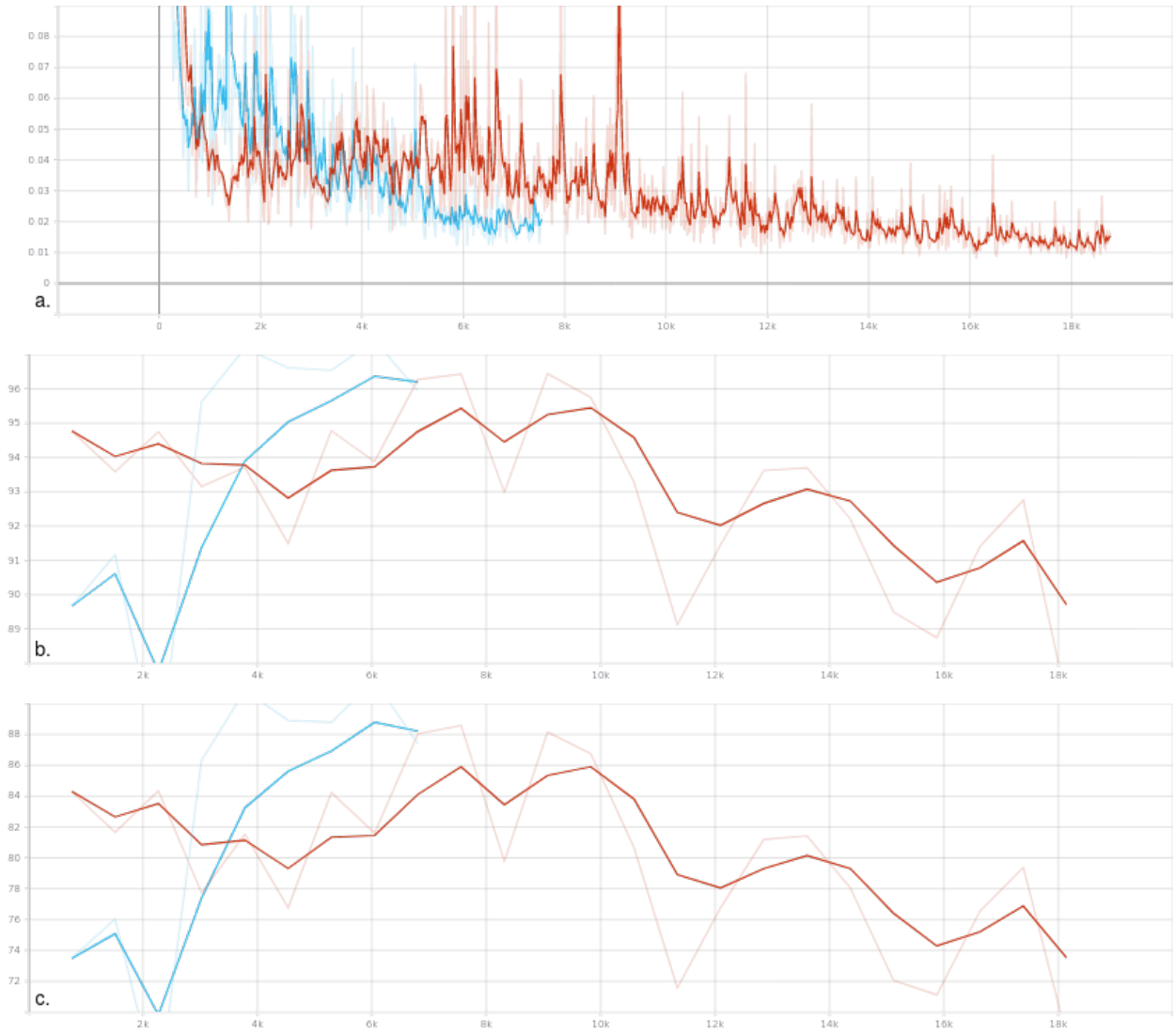


Рисунок 12 — Сравнение результатов one cycle policy и константого значения learning rate

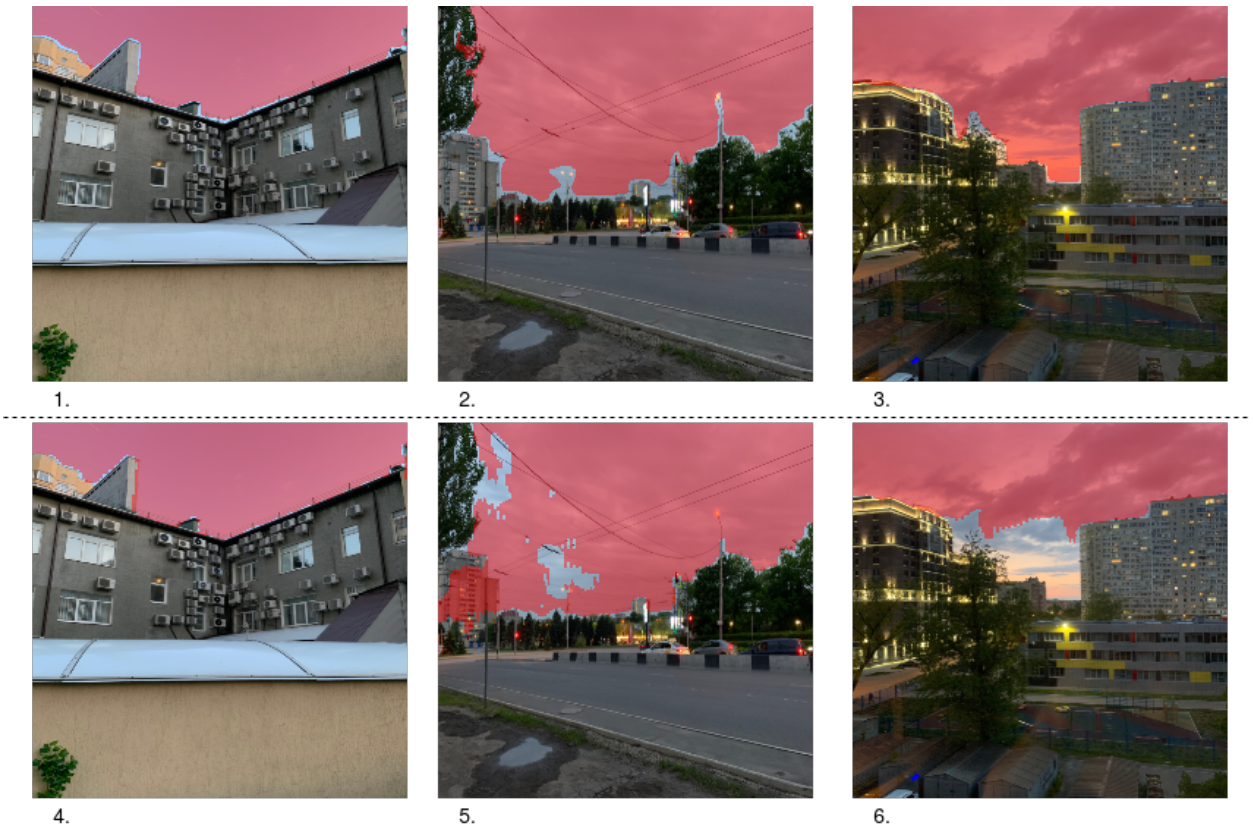


Рисунок 13 — Сравнение результатов для обучения с использованием синтетически размеченных данных (верхний ряд) с обучением только на SkyFinder

На 12 представлено сравнение техники one cycle policy настройки гиперпараметра (голубой) с обучением, в течение которого learning rate оставался постоянным (бурый). Как видно из графика, использование более высокого значения в первых итерациях позволило быстрее прийти к минимуму, а последующее уменьшение learning rate улучшило сходимость. На 12.a отображен график функции потерь, на 12.b и 12.c графики значений метрики IoU для классов неба и прочих объектов соответственно.

На 13 отражены результаты работы алгоритма сегментации. Красным цветом выделен сгенерированный сегмент неба. При использовании доменной адаптации (верхний ряд) можно видеть лучшее качество работы алгоритма с сегментами неба, в которых пред-

ставлено большее цветовое разнообразие. При этом подход без использования смены домена (нижний ряд) склонен выделять как ложноположительные сегменты неба при близости цветов (13.5) между объектами разных классов, так и оставлять ложноотрицательные регионы для резких переходов между цветам (13.6).

Заключение

В рамках данной работы был освещен вопрос семантического анализа фотографий искусственными нейронными сетями. В качестве предмета была выбрана задача сегментации пикселей изображения на два класса: принадлежащих области неба и относящихся к прочим объектам на снимке. Поставленная задача решалась применением глубоких сверточных сетей–ИНС со слоями свертки. Качество алгоритма было улучшено с помощью корректировки результирующей маски алгоритмами компьютерного зрения. При обучении сети использовались техники циклического изменения гиперпараметров learning rate и momentum, что также способствовало увеличению метрики IoU.

Для устранения влияния различий между качеством изображений в данных датасета SkyFinder и данных целевого домен, был применена доменная адаптация с помощью синтетически размеченных данных. Замечу, что использование изображений с искусственной разметкой показывает достаточно высокие результаты при низких временных затратах на подготовку данных.

Итоговое решение имеет средний показатель IoU между классами равный 94%. Сериализованная на физическом носители модель занимает $87.4\mathrm{MB}$.

A. Реализация сети с LinkNet декодером

```
1     class DecoderBlockLinkNet(nn.Module):
2         def __init__(self, in_channels, n_filters):
3             super().__init__()
4             self.relu = nn.ReLU(inplace=True)
5
6             self.conv1 = nn.Conv2d(in_channels, in_channels // 4, 1)
7             self.norm1 = nn.BatchNorm2d(in_channels // 4)
8
9             self.deconv2 = nn.ConvTranspose2d(in_channels // 4, in_channels //
                  4, kernel_size=4, stride=2, padding=1, output_padding=0)
10            self.norm2 = nn.BatchNorm2d(in_channels // 4)
11
12            self.conv3 = nn.Conv2d(in_channels // 4, n_filters, 1)
13            self.norm3 = nn.BatchNorm2d(n_filters)
14
15        def forward(self, x):
16            x = self.conv1(x)
17            x = self.norm1(x)
18            x = self.relu(x)
19            x = self.deconv2(x)
20            x = self.norm2(x)
21            x = self.relu(x)
22            x = self.conv3(x)
23            x = self.norm3(x)
24            x = self.relu(x)
25            return x
26
27
28    class LinkNet34(nn.Module):
29        def __init__(self, num_classes=1, num_channels=3, pretrained=True,
              debug_info=False,
30                     save_decoder_outputs=False, save_encoder_outputs=False):
31            super().__init__()
32            assert num_channels == 3
33            self.num_classes = num_classes
34            filters = [64, 128, 256, 512] #       -

35            resnet = models.resnet34(pretrained=pretrained)
36
37            self.firstconv = resnet.conv1
38            self.firstbn = resnet.bn1
39            self.firstrelu = resnet.relu
40            self.firstmaxpool = resnet.maxpool
41            self.encoder1 = resnet.layer1
```

```
42              self.encoder2 = resnet.layer2
43              self.encoder3 = resnet.layer3
44              self.encoder4 = resnet.layer4
45
46              self.decoder4 = DecoderBlockLinkNet(filters[3], filters[2])
47              self.decoder3 = DecoderBlockLinkNet(filters[2], filters[1])
48              self.decoder2 = DecoderBlockLinkNet(filters[1], filters[0])
49              self.decoder1 = DecoderBlockLinkNet(filters[0], filters[0])
50
51              self.finaldeconv1 = nn.ConvTranspose2d(filters[0], 32, 3, stride
                    =2)
52              self.finalrelu1 = nn.ReLU(inplace=True)
53              self.finalconv2 = nn.Conv2d(32, 32, 3)
54              self.finalrelu2 = nn.ReLU(inplace=True)
55              self.finalconv3 = nn.Conv2d(32, num_classes, 2, padding=1)
56
57              self.dropout1 = nn.Dropout2d(0.5, False)
58              self.dropout2 = nn.Dropout2d(0.3, False)
59
60              self.with_debug_info = debug_info
61              self.save_encoder = save_encoder_outputs
62              self.save_decoder = save_decoder_outputs
63
64          # noinspection PyCallingNonCallable
65          def forward(self, x):
66              x = self.firstconv(x)
67              x = self.firstbn(x)
68              x = self.firstrelu(x)
69              x = self.firstmaxpool(x)
70              e1 = self.encoder1(x)
71              e2 = self.encoder2(e1)
72              e3 = self.encoder3(e2)
73              e4 = self.encoder4(e3)
74
75              d4 = self.decoder4(e4) + e3
76              d3 = self.decoder3(d4) + e2
77              d2 = self.decoder2(d3) + e1
78              d1 = self.decoder1(d2)
79
80              f1 = self.finaldeconv1(d1)
81              f2 = self.finalrelu1(f1)
82              dr1 = self.dropout1(f2)
83              f3 = self.finalconv2(dr1)
84              f4 = self.finalrelu2(f3)
```

```
85              dr2 = self.dropout2(f4)
86              f5 = self.finalconv3(dr2)
87
88              x_out = F.log_softmax(f5, dim=1)
89
90              _ = self.with_debug_info and __print_info__(f5, x_out)
91              _ = self.save_encoder and __save_pre_encoder_outputs__([x])
92              _ = self.save_encoder and __save_encoder_outputs__([e1, e2, e3, e4
                    ])
93              _ = self.save_decoder and __save_decoder_outputs__([d1, d2, d3, d4
                    ])
94              _ = self.save_decoder and __save_classifier_outputs__([dr1])
95
96              return x_out
```

Строчки 1–25: Класс декодер-блока по схеме LinkNet. Функция forward (стр. 15) – прямой проход по сети. Библиотека глубокого обучения PyTorch, при использовании слоев из ее состава, позволяет не определять функции для обновления весов алгоритмам обратного распространения. В замен этого на каждой итерации обучения используются предописанные в самой библиотеке вычисления.

Строчки 28–96: Класс с полным описанием архитектуры сети.

Строчки 66–73: Исполнение енкодер-части сети.

Строчки 75–88: Исполнение декодер-части сети с последующей классификацией и применением результирующего Softmax.

Строчки 90–94: Снятие значений карт признаков с различных уровней сети и вывод отладочной информации. За счет применения механизма ленивого выполнения в языке Python, получается отказаться от оператора условного выбора, выполняемого за большое колво инструкций процессора

B. Реализация многоресурсного загрузчика

```
1       class SeveralSourceDataset(t_Dataset):
2       _modes = [
3           "absolute", "relative"
4       ]
5
```

```
6       def __init__(self, sources: Union[List[Union[Path, str]], Tuple[Union[Path
            , str]], Path, str],
7                    batch_consistency: Union[List[int], Tuple[int]],
8                    label_from_func: Callable, transforms: List[Any], size: int):
9           t = type(sources)
10          if t == Path or t == str:
11              sources = [sources]
12
13          self._sources: List[Path] = []
14          for s in sources:
15              if type(s) == str:
16                  self._sources.append(Path(s))
17              else:
18                  self._sources.append(s)
19
20          self._sources_count = len(self._sources)
21
22          self._bc = batch_consistency
23
24          if len(self._bc) != self._sources_count:
25              raise RuntimeError("Unexpected consistency value")
26
27          self._current_index = 0
28          self._index_to_source = []
29          index: int = 0
30          for bc in self._bc:
31              self._index_to_source += [index for _ in range(bc)]
32              index += 1
33
34          self.images: List[List[Path]] = []
35          self.labels: List[List[Path]] = []
36
37          self._len: int = 0
38          index: int = 0
39          for source in self._sources:
40              self.images.append([])
41              self.labels.append([])
42              for image in source.iterdir():
43                  self.images[index].append(image)
44                  self.labels[index].append(label_from_func(image))
45              self._len += len(self.images[index])
46              index += 1
47
```

```
48          self._inner_indexes: List[int] = [0 for _ in range(self._sources_count
                )]
49          if type(transforms) != list:
50              raise RuntimeError(
51                  "Unexpected transforms type. Should be list of fastai.vision.
                        image.RandTransform or list of list of fastai.vision.image
                        .RandTransform.")
52          if len(transforms) > 0:
53              if type(transforms[0]) == fastai.vision.image.RandTransform:
54                  transforms = [transforms]
55
56          self.transform: List[List[Any]] = transforms
57          self._size = size
58
59      def __len__(self) -> int:
60          return self._len
61
62      def get(self, item):
63          mod = len(self._index_to_source)
64          index = self._index_to_source[item % mod]
65          inner_index = self._inner_indexes[index]
66          img = open_image(self.images[index][inner_index])
67          label = open_mask(self.labels[index][inner_index], div=True)
68          inner_index += 1
69          self._inner_indexes[index] = inner_index % len(self.images[index])
70          return img, label
71
72      def __getitem__(self, item):
73          img, lbl = self.get(item)
74          img = img.resize(self._size).data
75          lbl = lbl.resize(self._size).data.squeeze(dim=0)
76          return img, lbl
```

Список литературы

1. *Gonzalez R. C.*, *Woods R. E.* Digital image processing. — Upper Saddle River, N.J. : Prentice Hall, 2008. — ISBN 9780131687288 013168728X 9780135052679 013505267X. — URL: http://www.amazon.com/Digital-Image-Processing-3rd-Edition/dp/013168728X.

2. *Liu X.*, *Deng Z.*, *Yang Y.* Recent progress in semantic image segmentation. — 2018. — arXiv: 1809.10198 [cs.CV].
3. *Wang C.*, *Ding J.*, *Chen P.* An efficient sky detection algorithm based on hybrid probability model // 2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA). — 2015. — с. 919—922.
4. A Gentle Introduction to Deep Learning in Medical Image Processing / A. Maier [и др.]. — 2018. — arXiv: 1810 . 05401 [cs.CV].
5. Image processing in DNA / C. Pan [и др.]. — 2019. — arXiv: 1910 . 10095 [eess.IV].
6. *Stabinger S.*, *Piater J.*, *Rodríguez-Sánchez A.* Evaluating the Progress of Deep Learning for Visual Relational Concepts. — 2020. — arXiv: 2001.10857 [cs.CV].
7. *Li K.* A Brief Survey of Image Processing Algorithms in Electrical Capacitance Tomography. — 2015. — arXiv: 1510.04585 [cs.CV].
8. Adaptive and Generic Corner Detection Based on the Accelerated Segment Test / E. Mair [и др.] // Proceedings of the 11th European Conference on Computer Vision: Part II. — Heraklion, Crete, Greece : Springer-Verlag, 2010. — с. 183—196. — (ECCV'10). — ISBN 3642155510.
9. *Hartigan J. A.* Clustering Algorithms. — 99th. — USA : John Wiley, Sons, Inc., 1975. — ISBN 047135645X.
10. Detecting People Using Mutually Consistent Poselet Activations / L. Bourdev [и др.] //. — 12.2010. — с. 168—181. — DOI: 10.1007/978-3-642-15567-3_13.
11. *Ogiela M.*, *Tadeusiewicz R.* Modern Computational Intelligence Methods for the Interpretation of Medical Images. т. 84. —

01.2008. — с. 91. — ISBN 978-3-540-75399-5. — DOI: 10.1007/978-3-540-75402-2.

12. Comparison of semantic segmentation approaches for horizon/sky line detection / T. Ahmad [и др.] // 2017 International Joint Conference on Neural Networks (IJCNN). — 2017. — с. 4436—4443.

13. Deep Multi-modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges / D. Feng [и др.]. — 2019. — arXiv: 1902.07830 [cs.RO].

14. *Ronneberger O.*, *Fischer P.*, *Brox T.* U-Net: Convolutional Networks for Biomedical Image Segmentation. — 2015. — arXiv: 1505.04597 [cs.CV].

15. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs / L.-C. Chen [и др.]. — 2016. — arXiv: 1606.00915 [cs.CV].

16. RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation / G. Lin [и др.]. — 2016. — arXiv: 1611.06612 [cs.CV].

17. *Place C. L.*, *Khan A. U.*, *Borji A.* Segmenting Sky Pixels in Images. — 2017. — arXiv: 1712.09161 [cs.CV].

18. *Salman S.*, *Liu X.* Overfitting Mechanism and Avoidance in Deep Neural Networks. — 2019. — arXiv: 1901.06566 [cs.LG].

19. *Ghojogh B.*, *Crowley M.* The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial. — 2019. — arXiv: 1905.12787 [stat.ML].

20. *Smith L. N.* A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. — 2018. — arXiv: 1803.09820 [cs.LG].

21. *Labach A.*, *Salehinejad H.*, *Valaee S.* Survey of Dropout Methods for Deep Neural Networks. — 2019. — arXiv: 1904.13310 [cs.NE].

22. *Ioffe S.*, *Szegedy C.* Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. — 2015. — arXiv: 1502.03167 [cs.LG].

23. *Dumoulin V.*, *Visin F.* A guide to convolution arithmetic for deep learning. — 2016. — arXiv: 1603.07285 [stat.ML].

24. *Shimodaira H.* Improving predictive inference under covariate shift by weighting the log-likelihood function // Journal of Statistical Planning and Inference. — 2000. — окт. — т. 90. — с. 227—244. — DOI: 10.1016/S0378-3758(00)00115-4.

25. Deep Residual Learning for Image Recognition / K. He [и др.]. — 2015. — arXiv: 1512.03385 [cs.CV].

26. *Chaurasia A.*, *Culurciello E.* LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation. — 2017. — arXiv: 1707.03718 [cs.CV].

27. Demystifying Learning Rate Policies for High Accuracy Training of Deep Neural Networks / Y. Wu [и др.]. — 2019. — arXiv: 1908.06477 [cs.LG].

28. *Suzuki S.*, *Abe K.* Topological structural analysis of digitized binary images by border following // Comput. Vis. Graph. Image Process. — 1985. — т. 30. — с. 32—46.

29. Sky Segmentation in the Wild: An Empirical Study / R. P. Mihail [и др.] // IEEE Winter Conference on Applications of Computer Vision (WACV). — 2016. — с. 1—6.

30. Improving Semantic Segmentation via Self-Training / Y. Zhu [и др.]. — 2020. — arXiv: 2004.14960 [cs.CV].