Санкт-Петербургский Государственный Университет

ПАВЛОВ Илья Иванович

Выпускная квалификационная работа

Демографический анализ спутниковых изображений с использованием нейронных сетей

Магистр:
Направление *02.04.02 «Фундаментальная информатика и информационные технологии»*
Основная образовательная программа *ВМ.5786*. «Цифровые технологии и системы»

Научный руководитель:
профессор, кафедра компьютерных технологий и систем, доктор физ.-мат. наук, Сотникова М. В.

Санкт-Петербург

2021

ОГЛАВЛЕНИЕ

Введение 2
Глава 1. Постановка задачи 4
1.1. Формулировка задачи демографического анализа 4
1.2. Математическая постановка задачи 5
1.3. Обзор литературы 8
Глава 2. Описание Алгоритма 11
2.1 Обзор данных 11
2.2 Панхроматическое слияние 16
2.3. Семантическая сегментация 18
2.4. Прогнозирование численности населения 21
2.5. Улучшение прогноза численности населения 23
Глава 3. Реализация и результаты 24
3.1. Реализация модифицированной сети U-Net++ 24
3.2. Реализация сети прогнозирования диапазона численности населения 32
3.3. Реализация регрессионных моделей 40
Заключение 43
Список литературы. 44
Приложение 47

Введение

Актуальность темы исследования. Точное знание численности населения является критичной информацией для таких задач, как градостроение, планирование борьбы с инфекционными заболеваниями, уменьшение ущерба от природных катастроф и социально-экономических исследований.

На текущий момент единственный надёжный способ получения информации о фактической численности населения является перепись. Этот процесс невероятно трудоёмкий и дорогой. Даже с автоматизацией и цифровизацией некоторых процессов, перепись занимает месяцы и требует отправлять переписчиков в труднодоступные районы без постоянного транспортного сообщения и надёжных способов коммуникации.

Поскольку переписи, как правило проходят раз в 10 лет, то если для решения задачи требуется текущая численность населения некоторой территории, но единственные доступные данные были собраны несколько лет назад, это может существенно затруднить её выполнение. Вдобавок к этому если между переписями произошли события, которые могли сильно и за короткое время изменить число населения рассматриваемой территории, данные переписи могут стать вообще бесполезными.

На текущий момент на земной орбите находятся более чем 150 спутников, которые способны брать высококачественные снимки поверхности Земли в момент, когда это нам потребуется. Именно эта их способность может помочь решить проблему потери актуальности данных переписи.

Ухудшающаяся экологическая ситуация и следующие из этого природные катастрофы и социально-экономические кризисы делает задачу создания системы, которая будет способна получать релевантные данные о численности населения, используя только спутниковые изображения, как никогда актуальной.

Цели и задачи. Целью работы является изучение применения сверточных нейронных сетей для демографического анализа спутниковых изображений.

Задачами этой работы являются:

- Создание и тренировка нейронной сети для семантической сегментации спутниковых снимков на типы земных покровов.
- Провести ряд экспериментов с разными модификациями нейронной сети для семантической сегментации.
- Создание и тренировка сетей для прогнозирования численности населения со спутниковых снимков.
- Эмпирически доказать, что дополнение спутниковых изображений картами земных покровов, которые генерируются нейронными сетями семантической сегментации, приведет к улучшению способности моделей прогнозировать численность населения.

Структура. Работа состоит из введения, трех глав, заключения, списка литературы и приложения.

Глава 1. Постановка задачи

1.1. Формулировка задачи демографического анализа

На текущий момент системы способные прогнозировать число населения со спутниковых снимков страдают от одной проблемы, они затрудняются давать корректные результаты для плотно застроенных городских территорий. Предложенный в данной работе способ решения этой проблемы является применение второй глубокой нейронной сети для генерации дополнительной информации об изображении, а именно выделение разных типов земного покрова на изображении. Этот метод похож на добавление инфракрасных и термальных каналов к RGB изображениям, но главным отличием и преимуществом является более четкое разделение городов на разные районы. Кроме этого, интегрирование этих двух моделей в одну систему даст инструмент, который способен возвращать более крупное количество информации и дополнительный контекст для размера населения.

Использование карт земных покровов дает дополнительную цель для этой работы. В процессе выполнения будет получено эмпирического доказательство, что изображения, подкрепленные каналом земного покрова, помогают нейронным сетям лучше прогнозировать размер населения или нет.

В качестве входных данных для этой системы будут использоваться изображения, вырезанные из спутниковых фотографий городов, лесов, океанов и прочих типов территорий. Изображения должны быть достаточно маленькими из-за ограничений аппаратного обеспечения.

Результатом работы первой нейронной сети будет матрица, где каждая ячейка соответствует определенному типу земного покрова. Эта матрица будет добавлена к начальному изображению в качестве дополнительного канала. Это расширенное изображение будет отправлено на вход второй нейронной сети, которая, в свою очередь, вернет номер некоторого класса диапазона населения.

Задачей второй сети является классификация. Регрессия на этом шаге не используется, из-за крупного диапазона населения на территории, представленной на вырезанных изображениях (от 0 до сотен тысяч), что приводит к крайней неточности и трудностями при ее обучении.

Финальным шагом является улучшение результатов, используя регрессионные модели, которые ограниченны диапазонами, полученными на предыдущем шаге.

Требования к точности работы этой системы высоки. Система будет непригодной, если она не сможет различать между лесом или центром города, но также если она не сможет различать и корректно прогнозировать размер населения разных типов территорий. В дополнении к этому система должна успешно работать как и в масштабе городского района, так и в масштабе страны.

Практическое исполнение системы было реализовано с помощью языка программирования Python 3, библиотеки алгоритмов компьютерного зрения Opencv 3, библиотек машинного обучения Tensorflow 2 и scikit-learn, сервиса облачных вычислений Google Colab. Для работы с геопространственными форматами данных GeoTIFF была использована библиотека GDAL и система QGis 3.14.

1.2. Математическая постановка задачи

В работе требуется построить оператор A, способный получая на вход изображение I вывести число P, представляющее численность населения на I.

Входными данными для оператора являются изображения произвольного размера в цветной модели RGB, которые представлены трехмерной матрицей $W \times H \times 3$. Где $W-$ ширина изображения, а $H-$ высота изображения.

Предложенный оператор A состоит из трех частей: сегментатор, классификатор и регрессионной модели.

Целью сегментатора является, используя RGB изображение I получить на выход карту сегментации S с размерностью $W \times H \times 1$, где каждый пиксель равен

целому значению одного из заранее известных категорий. Получив эту карту, она добавляется к начальному изображению I в качестве дополнительного канала. Таким образом, на выходе сегментатора получаем тензор I_{aug}, размерностью $W \times H \times 4$.

Для подсчета качества работы сегментатора применяется метрика Mean Intersection Over Union (1.1). Эта метрика вычисляет среднее значение сходства между всеми вычисленными семантическими классами и их действительными значениями:

$$meanIoU = \sum_{i=1}^{n} \frac{TP_i}{TP_i + FP_i + FN_i}, \quad (1.1)$$

где n – количество семантических классов,

TP_i – доля истинно положительных значений,

FP_i – доля ложно положительных значений,

FN_i – доля ложно негативных значений.

Целью классификатора является, используя тензор I_{aug} получить значение номера одного из диапазонов численности населения, которое проживает на территории, представленной изображением I.

Точность работы классификатора будет измеряться с помощью метрики Accuracy (1.2):

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}, \quad (1.2)$$

где $TN -$ доля истинно негативных значений.

Целью регрессионной модели является, используя вектор X получить значение число населения, которое проживает на территории, представленной изображением I.

Точность работы будет измеряться с метрики Mean Absolute Error (MAE) (1.3):

$$MAE = \frac{\sum_{i=1}^{N} |y_i - x_i|}{N}, \quad (1.3)$$

где $y_i -$ полученное значение,

x_i – настоящее значение,

N – общее число рассматриваемых значений.

Схема предложенной системы показана на рисунке 1.1.

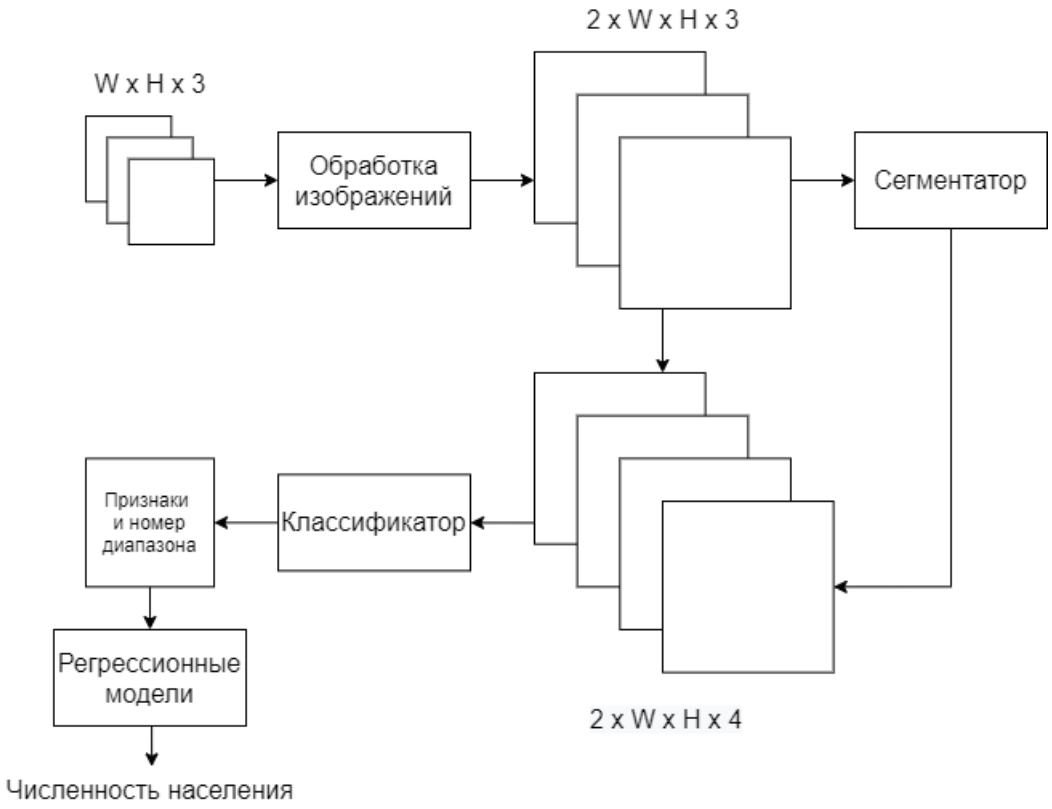

Рисунок 1.1 – Схема предложенной в этой работе системы

Спутниковые изображения, используемые для тренировки обеих нейронных сетей, были взяты из набора снимков, снятых спутником LandSat-7 [1] в 2000 году. Этот год был выбран по причине наименьшего количества артефактов на снимках. Сами изображения имеют 7 каналов, но использовались только первые три:

1. Красный (0,45–0,52 мкм), с разрешением 30 м^2.
2. Зеленый (0,52–0,60 мкм) с разрешением 30 м^2.
3. Синий (0,63–0,69 мкм) с разрешением 30 м^2.
4. Ближний инфракрасный (0,77–0,90 мкм) с разрешением 30 м^2.
5. Коротковолновый инфракрасный (1,55–1,75 мкм) с разрешением 30 м^2.
6. Термальный (10,40–12,50 мкм) с разрешением 60 м^2.

7. Средне инфракрасный (2,08–2,35 мкм) с разрешением 30 $м^2$.
8. Панхроматический (0,52–0,90 мкм) с разрешением 15 $м^2$.

Всего было взято 18 снимков городов на всей территории континентальной США. Снимки были разделены на множество изображений размерном $67 \times 67 \times 3$, что представляет собой территорию 2×2 километра. В итоге для тренировки и тестирования было получено 140000 изображений, количество которых было увеличено после процесса расширения данных. Для обработки снимков был применен метод панхроматического слияния [2], который, используя информацию из панхроматического сенсора, увеличивает размерность изображения в 2 раза.

Данные для семантических классов были взяты из набора маркированных данных земного покрова из набора NLCD [3] предоставленные Геологической службой США. Они представляют собой матрицу, где каждая ячейка соответствует номеру класса покрова земли в квадрате размером 30 $м^2$.

Данные о численности населения были взяты из набора U.S. Census Grid [4] предоставленные SEDAC. Они представляют из себя матрицу, где каждая ячейка соответствует взвешенному числу населения в квадрате размером 1 $км^2$.

1.3. Обзор литературы

Задача автоматизированного анализа спутниковых снимков стала актуальной с того момента, как первые спутники способные брать фотографии Земли были запущены на орбиту. Ранние подходы к задаче классификации применяющие такие классические методы, как Local Binary Descriptors [5], SIFT [6], HOG [7], SVM [8], показали себя довольно эффективными. Из-за высокой размерности полученных пространств признаков требовался перевод их в более компактный вид, и поэтому большое количество научных работ было посвящено именно разным методам кодирования этих данных [9], [10].

Глубокое обучение доказало себя, как надежный метод решения широкого спектра задач. Оно и нашло свое применение в анализе спутниковых снимков.

Одно из первых исследований применения нейронных сетей в этой задаче рассматривало задачу поиска дорожных перекрестков [11]. После того как сверточные нейронные сети начали набирать популярность в середине 2010-х годов начали выходить статьи, исследующие возможность их использования в анализе спутниковых изображений [12], [13]. В этих ранних работах сверточные нейронные сети использовались только для генерации вектора признаков, который далее использовался для тренировки методов погожих на методы в [7-8].

На текущий момент, тематика исследования спутниковых изображений с помощью сверточных нейронных сетей хоть и не является самой популярной, но тем не менее на эту тему было написано достаточно большое количество научных работ. Большинство из них озабочены классификацией и сегментацией участков спутниковых снимков или улучшением их качества. Примеры использования глубокого обучения включают: прогнозирование размера населения [14], сегментирование изображений [15], прогнозирование уровня бедности [16], панхроматическое слияние [17].

Самое быстро развивающееся направление исследований является использование генеративно-состязательных нейронных сетей для генерации абсолютно новых изображений, которые сложно отличить от настоящих невооруженному глазу [18], [19].

Для тренировки любой нейронной сети требуются данные и по причине вездесущности продуктов, использующих спутниковые изображения в повседневной жизни, можно найти огромное их количество. В качестве примеров можно привести такие наборы, как снимки, снятые спутниками Landsat-7, Landsat-8, Sentinel-2 и наборы изображений DeepSat [20].

Методика сбора данных и общий подход к задаче прогнозирования населения с помощью сверточных нейронных сетей были вдохновлены работой [14]. Исследователи использовали данные полученные от бюро переписи США с 2000 и 2010 годов. Для спутниковых изображений были использованы наборы, полученные от спутника Landsat-7. Каждое изображение состояло из 8 диапазонов: красный, синий, зеленый, два диапазона из ближней инфракрасной области, тер-

мический, средняя инфракрасная область и панхроматическая. В качестве архитектуры глубокой нейронной сети были использованы, уже к тому времени устаревшая, укороченная сеть VGG-16. Их конечные результаты подошли близко к традиционным методам прогнозирования населения, которые использует бюро переписи США.

Семантическая сегментация другая активно развивающаяся направление в мире глубокого обучения. Она применяется в задачах обнаружения объектов [21], беспилотного пилотирования [22] и анализа медицинских изображений [23]. Существует множество архитектур для этой задачи, самыми популярными из которых являются FCN [24] и U-Net [25]. В частности, архитектура нейронной сети для работы с медицинскими изображениями U-Net была модифицирована большое количество раз и приспособлена для широко спектра задач: Attention U-Net [26], U-Net GAN [27], U-Net++ [28] и большое количество других.

Идея сегментирования спутниковых изображений была впервые представлена в работе [29]. В ней исследователи применили такие глубокие сверточные нейронные сети как CaffeNet и GoogleLeNet, с достаточно удовлетворительными результатами. Работа [30] сравнила разные более актуальные модели глубоких нейронных сетей. В итоге исследователи пришли к выводу, что все модели они имеют проблемы с сегментацией районов со средней и высокой плотностью застройки, но малонаселенные районы не предоставляли им трудности.

Глава 2. Описание Алгоритма

2.1 Обзор данных

Для тренировки и тестирования всех нейронных сетей, задействованных в системе, требуется большое количество данных из разных источников. Нам потребуются многоканальные спутниковые изображения, данные о численности населения и размеченные карты земного покрова.

Спутниковые изображения, используемые в работе, были сняты спутником LandSat-7 с марта по октябрь 2000-го года. В набор данных только брались изображения без облачного покрова. Каждый снимок охватывал территорию размером примерно 185 км^2. Описания каналов изображений были представлены в первой главе. Всего было использовано 19 снимков самых крупно населенных городов Соединённых Штатов Америки. Для тренировки нейронных сетей использовались вырезки из снимков размером 67×67 пикселей, что примерно представляет территорию размером 2 км^2. Примеры этих изображений показаны на рисунках 2.1 и 2.2.

Рисунок 2.1 – Примеры спутниковых снимков, снятых спутником LandSat-7

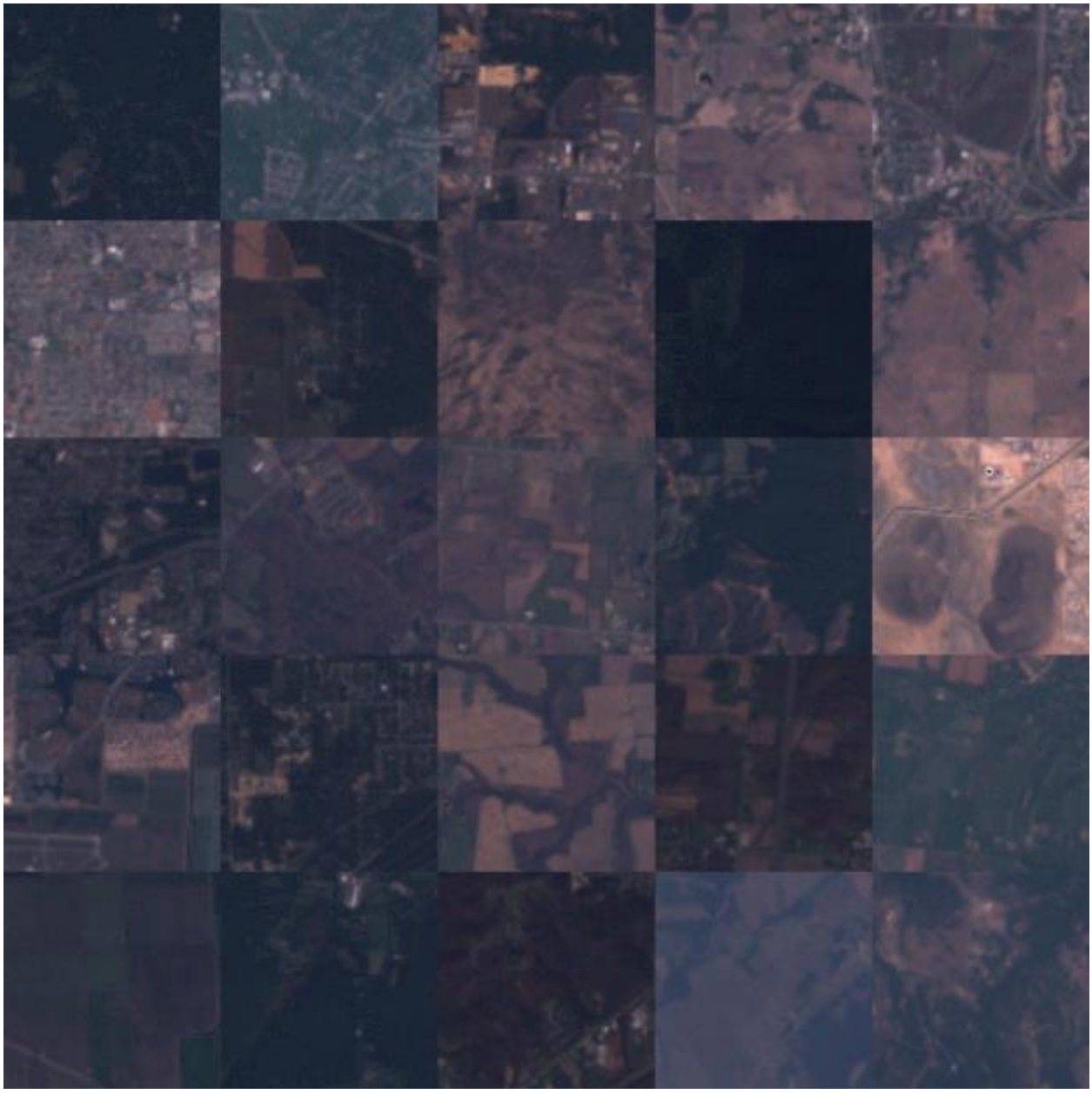

Рисунок 2.2 – Примеры вырезок из спутниковых снимков, используемые для тренировки нейронных сетей

Данные о численности населения были взяты из набора данных, собранного используя результаты переписи США 2000-го года. Данные представляют из себя матрицу с разрешением приблизительно 1 км^2 и покрывает всю территорию континентальных США. Поскольку каждое изображение, используемое для тренировки, покрывает территорию размером 2 км^2, численность для каждого изображения вычисляется путем усреднения значений. Для тренировки нейронной сети

прогнозирования диапазона численности населения, численность населения P было представлено 11-ью диапазонами:

1. $P = 0$
2. $P \in (0, 15]$
3. $P \in (15, 40]$
4. $P \in (40, 100]$
5. $P \in (100, 200]$
6. $P \in (200, 500]$
7. $P \in (500, 800]$
8. $P \in (800, 2000]$
9. $P \in (2000, 4000]$
10. $P \in (4000, 6000]$
11. $P > 6000$

На рисунке 2.3 продемонстрирована гистограмма диапазонов населения.

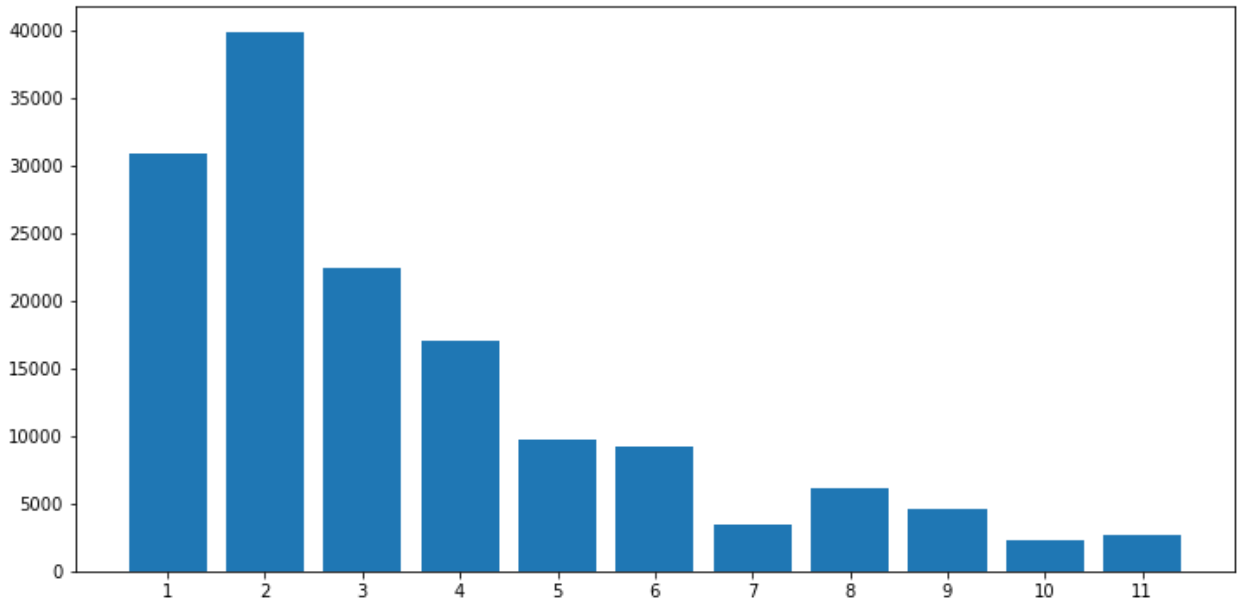

Рисунок 2.3 – Гистограмма диапазонов населения

Используя этот набор данных, можно получить численность населения для каждого из рассматриваемых спутниковых снимков:

- Атланта 14.10.2000, 2.536 млн.
- Бостон 27.09.2000, 4.115 млн.
- Чикаго 10.10.2000, 5.927 млн.
- Колумбус 30.10.2000, 1.964 млн.
- Даллас 04.09.2000, 3.732 млн.
- Хьюстон 06.09.2000, 3.298 млн.

- Индианаполис 06.06.2000, 1.743 млн.
- Канзас-Сити 13.04.2000, 1.373 млн.
- Лос-Анджелес 06.09.2000, 9.145 млн.
- Мемфис 10.10.2000, 9.607 млн.
- Майями 05.02.2000, 3.589 млн.
- Миннеаполис 29.04.2000, 2.025 млн.
- Нью-Йорк 20.10.2000, 1.111 млн.
- Филадельфия 04.05.2000, 4.913 млн.
- Феникс 19.04.2000, 2.25 млн.
- Сиэтл 25.09.2000, 2.09 млн.
- Сан-Франциско 03.03.2000, 5.071 млн.
- Вашингтон 24.03.2000, 5.075 млн.

Размеченные карты земного покрова были взяты из набора данных NLCD Land Cover (CONUS) за 2001 год. Данные представляют из себя матрицу с разрешением приблизительно 30 $м^2$ и состоят из 20 классов. В эти классы входят: вода, многолетний лёд, застроенные и бесплодные территории, разные типы лесов, территорий, покрытых кустарниками, травянистые и культивированные покровы и водно-болотные территории. Для нашей задачи нам потребуются только шесть классов земных покровов. Ниже перечислены эти классы и цвета, которыми они будут представлены на изображениях:

1. Вода, синий.
2. Крайне низкая плотность застройки, серый.
3. Низкая плотность застройки, оранжевый.
4. Умеренная плотность застройки, красный.
5. Высокая плотность застройки, фиолетовый.
6. Все остальное (все ненаселенные и незастроенные территории объединены в один класс), темно-коричневый.

На рисунке 2.4 пример карты земного покрова из набора NLCD, а на рисунке 2.5 земные покровы снимков на рисунке 2.2.

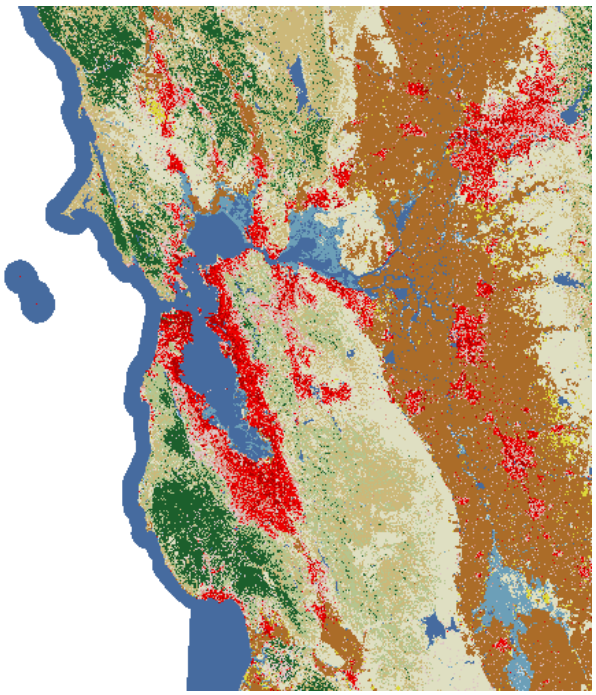

Рисунок 2.4 – Пример данных из набора NLCD

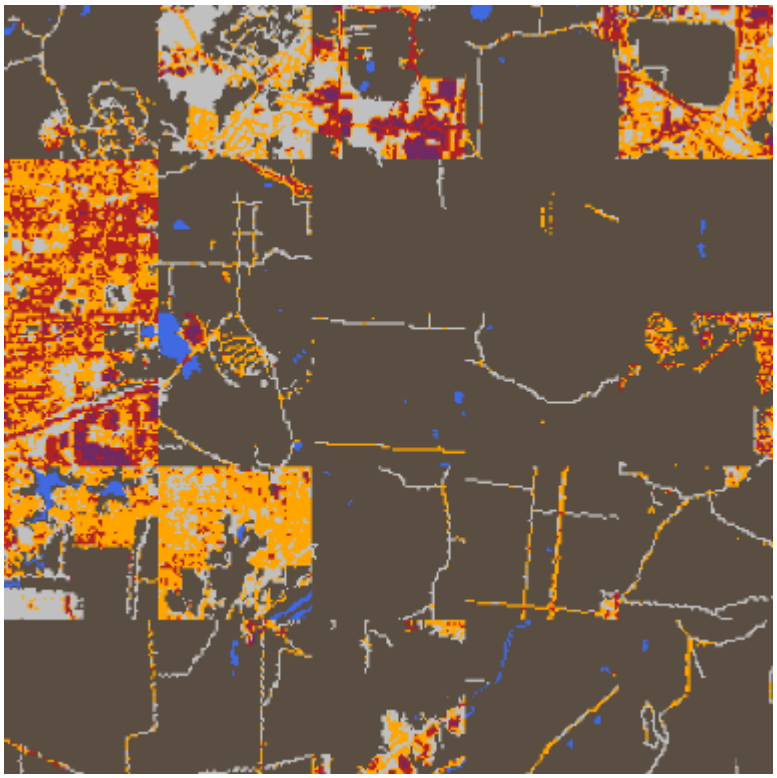

Рисунок 2.5 – Примеры вырезок из карт земных покровов

2.2 Панхроматическое слияние

Панхроматическое слияние использовалось в процессе предобработки изображений. Этот метод объединяет панхроматический канал спутникового снимка с каналами более низкого разрешения с целью увеличения их размерности, что позволяет увеличить количество данных для тренировки последующих нейронных сетей и увеличить количество деталей на них. В этой работе использовался метод панхроматического слияния, который был предложен в статье [2]. На рисунке 2.6 показан пример результата работы этого алгоритма.

Данный алгоритм предлагает вычислить матрицу V, которую при слиянии данных, полученных из разных сенсоров, будет выполняться равенство (2.1):

$$W * S + V = P, \tag{2.1}$$

где S – энергия данных спектрального сенсора,

W – веса, вычисленные для каждого канала спектрального сенсора,

P – энергия данных панхроматического сенсора.

Алгоритм берет на вход S_{lr} − мультиканальное изображение низкого разрешения и P_{hr} − панхроматическое изображение до размера, которого мы хотим увеличить изображение S_{lr}.

Веса W аппроксимируются вектором $\widehat{W}$ полученный минимизацией формулы (2.2) используя алгоритм минимизации BVLS с ограничениями на веса $0 \leq w \leq 1$:

$$W * S_{lr} = P_{lr}, \tag{2.2}$$

где P_{lr} – панхроматическое изображение, уменьшенное до размера изображения S_{lr}.

Виртуальный канал V аппроксимируется матрицей V_{hr}, который получается в результате бикубической интерполяции матрицы V_{lr} полученной формулой (2.3):

$$V_{lr} = P_{lr} - \widehat{W} * S_{lr}. \tag{2.3}$$

Используя V_{hr}, производится корректировка изображения P_{hr} по формуле (2.4):

$$\widehat{P_{hr}} = P_{hr} - V_{hr}. \quad (2.4)$$

Полученная матрица $\widehat{P_{hr}}$ будет соблюдать равенство (2.1).

Финальным шагом алгоритма является слияние S_{hr} и $\widehat{P_{hr}}$ для получения финального изображения $\widehat{S_{hr}}$ (2.5-2.6):

$$\widehat{I_{hr}} = \widehat{W} * S_{hr}, \quad (2.5)$$

$$\widehat{S_{hr}} = S_{hr} + \widehat{P_{hr}} - \widehat{I_{hr}}. \quad (2.6)$$

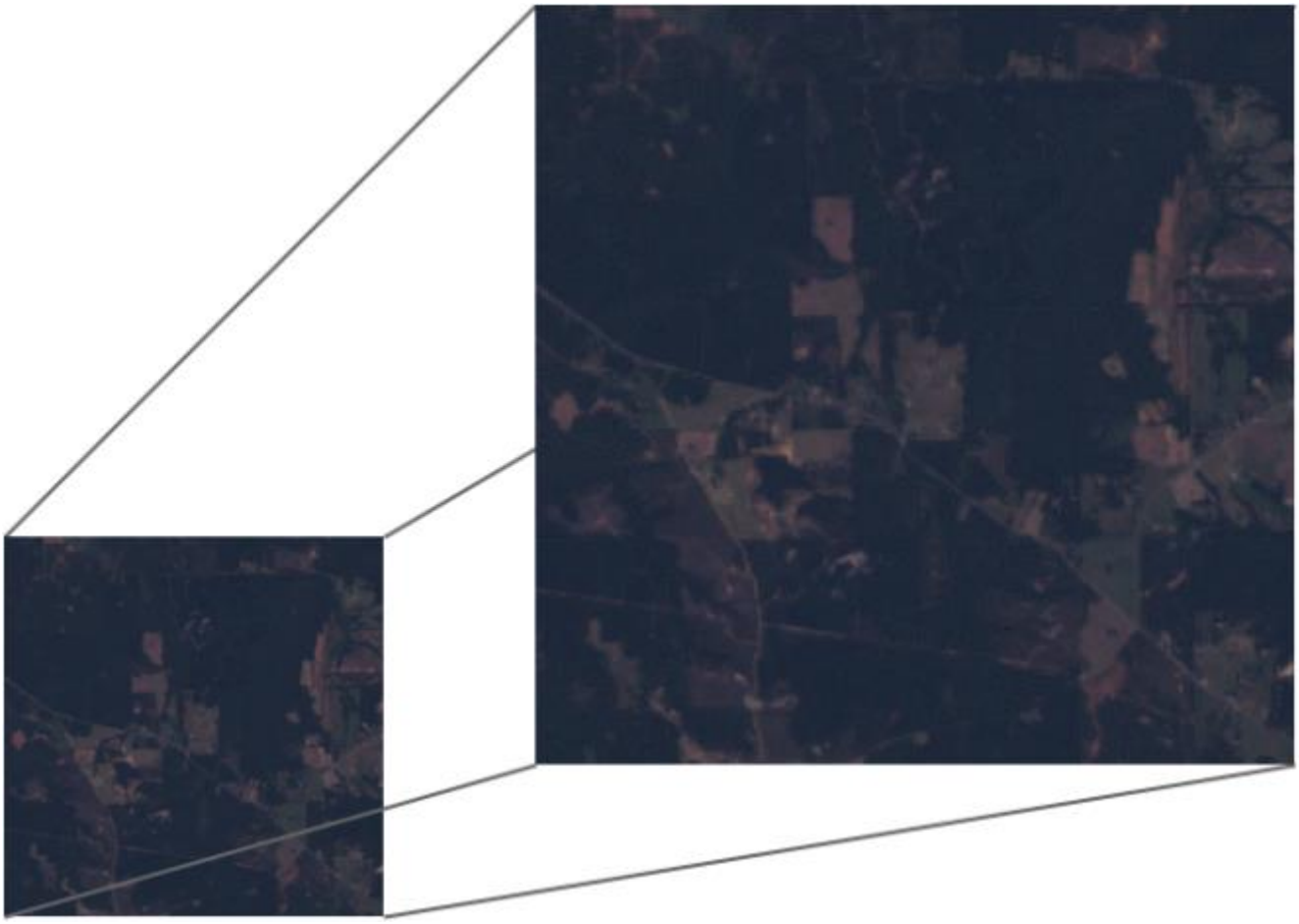

Рисунок 2.6 – Пример результата работы алгоритма панхроматического слияния

2.3. Семантическая сегментация

Задачей семантической сегментации является по пиксельная классификация входного изображения. Для этого нейронным сетям требуется реконструировать данные, так чтобы их высота и ширина совпадала с высотой и шириной входного изображения.

Архитектура нейронной сети U-Net++ [28] была создана для задачи семантической сегментации медицинских изображений и является развитием идей позади архитекторы нейронной сети U-Net. Эта архитектора была выбрана по той причине, что для сегментации спутниковых снимков городов требуется совершать операции похожие на те, которые требуется делать для сегментации медицинских изображений.

Сеть U-Net [25] состоит из двух частей: энкодера и декодера. Во время прохода данных через энкодер входное изображение постепенно преобразуется в вектор. Этот вектор отправляется в декодер, где он постепенно увеличивается в размере и совмещается с данными, которые были сохранены во время прохода данных через энкодер, это позволяет сохранить целостность данных, что уменьшает искажения во время реконструкции. В результате данные возвращаются к начальному размеру, готовыми для классификации. Схема архитектуры U-Net показана на рисунке 2.7.

Энкодер сжимает входные изображения используя повторяющиеся блоки из двух сверточных слоев с окном свертки 3×3 и постепенно увеличивающимися количествами фильтров (2.7). После каждого слоя следует функция активации ReLU (2.8), а после каждого блока операция уменьшения размерности карты признаков, Max Pooling с окном 2×2 и шагом 2 (2.9).

$$C_{i,j} = \sum_{u=0}^{m_x-1} \sum_{v=0}^{m_y-1} A_{i+u,j+v} B_{u,v}\,, \tag{2.7}$$

$$C_{i,j_{ReLU}} = \begin{cases} 0, C_{i,j} < 0 \\ C_{i,j}, C_{i,j} \geq 0' \end{cases} \tag{2.8}$$

$$D_{i_1,j_1} = \max_{c \in X_{k \times l}} c\,, \tag{2.9}$$

где матрица A – входные данные в блок,

B – окно свертки размера $m_x \times m_y$,

X – регион $C_{i,j_{ReLU}}$ размером $k \times l$.

Декодер состоит из повторяющихся блоков, которые состоят из:

- Операция интерполяции методом ближайшего соседа, увеличивающая размерность данных.
- Сверточного слоя с окном свертки 2×2, функцией активации ReLU и постепенно уменьшающимся количеством фильтров.
- Пропускного соединения, которое выполняет операцию конкатенацию данных с соответствующими по размеру данными из энкодера.
- Двух сверточных слоев с окном свертки 3×3 и функцией активации ReLU.
- Финальный слой сети состоит из сверточного слоя с окном 1×1, для того чтобы соединить каждый из 64 каналов данных с вектором, чей размер равен количеству семантических классов, которых надо сегментировать.

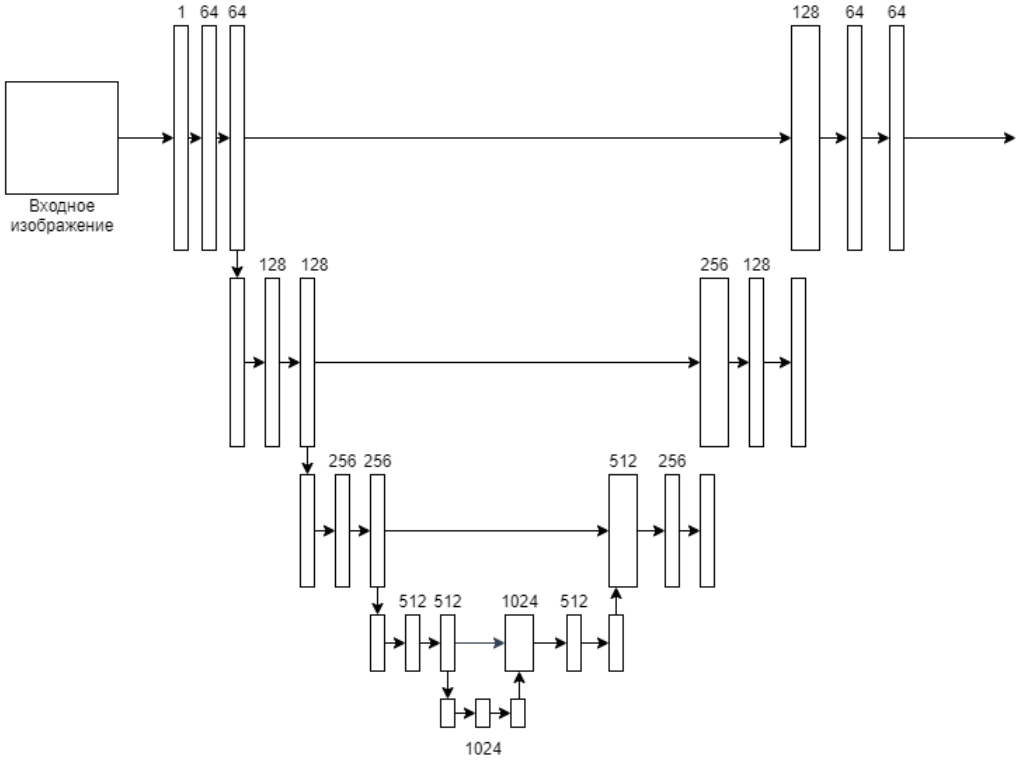

Рисунок 2.7 – Архитектура сети U-Net

Нейронная сеть U-Net++ имеет три нововведения по сравнению с U-Net:

1. Плотные пропускные соединения.
2. Новые пропускные пути.
3. Deep supervision [31].

Плотные пропускные соединения были взяты из нейронной сети DenseNet [32]. Эти соединения аккумулируют все данные предыдущих блоков перед тем, как отправить их в текущий, что увеличивает качество работы нейронной сети и облегчает вычисление градиента во время тренировки.

Новые пропускные пути устраняют информационный разрыв между данными энкодера и декодера, что облегчает проблему оптимизации во время обучения нейронной сети. Они представляют собой обучаемые сверточные блоки с окном свертки 3×3, плотными пропускными соединениями и дополнительно конкатенируют к своим входным данным увеличенные в размерности данные из блока на уровень ниже.

Архитектура сети позволяет иметь несколько выходов, которые отправляют на выход данные с размерностью $W \times H \times F$, где W, H – ширина и высота входного изображения, а F количество фильтров на первом уровне сети. Прием Deep Supervision усредняет данные из этих выходов для получения финального результата. Этот метод также допускает использовать любую комбинации выходов, что дает выбор между скоростью и качеством работы нейронной сети.

Сеть U-Net++ использует следующую функцию потери (2.10):

$$\mathrm{L}(y_{true}, y_{pred}) = -\frac{1}{N}\sum_{c=1}^{C}\sum_{n=1}^{N}\left(y_{true_{n,c}} log y_{pred_{n,c}} + \frac{2y_{true_{n,c}}y_{pred_{n,c}}}{y_{true_{n,c}}^2 + y_{pred_{n,c}}^2}\right), \qquad (2.10)$$

где y_{true} – корректные данные,

y_{pred} – спрогнозированные данные,

C – число классов,

N – число пикселей.

В результате архитектура нейронной сети U-Net++ не только дает более точные результаты семантической сегментации, но и является более гибкой сетью по сравнению с оригинальной сетью U-Net. Графическое представление архитектуры этой сети показано на рисунке 2.8. Зеленными линиями обозначены операции увеличения размерности, черными пунктирными линиями обозначены пропускные соединения, оставленные из сети U-Net, синими пунктирными линиями обозначены плотные пропускные соединения, новые пропускные пути обозначены зелеными кругами.

Рисунок 2.8 – Архитектура нейронной сети U-Net++

2.4. Прогнозирование численности населения

Для выполнения задачи прогнозирования диапазона численности населения предложена архитектура нейронной сети, продемонстрированная на рисунке 2.10. Нейронная сеть берет на вход тензор, состоящий из четырех каналов: красный, зелёный, синий и матрица семантически сегментированного изображения. Сеть ветвится на две части, обе состоящие из архитектуры нейронной сети ResNet [33]. В конце обе сети объединяются одним сверточным слоем.

Параллельная архитектура использовалась из-за гетерогенности между RGB и семантически сегментированным изображением. Когда, как RGB изображение напрямую представляет сфотографированную территорию, то семантически сегментированного изображение является результатом работы нейронной сети, которая только аппроксимирует настоящую информацию.

Архитектуры сетей семейства ResNet используют остаточные блоки. На Рисунке 2.9 продемонстрирована разница между обычными и остаточными сверточными блоками. Слева целью обучения является превращение выделенной части блока в функцию $f(x)$, а справа в функцию $f(x) - x$. Если требуемая функция $f(x)$ является тождественным отображением $f(x) = x$, то правому блоку достаточно назначить весам в первом сверточном слое нулевые значения. Этот процесс позволяет сильно облегчить задачу оптимизации и увеличить качество работы нейронных сетей.

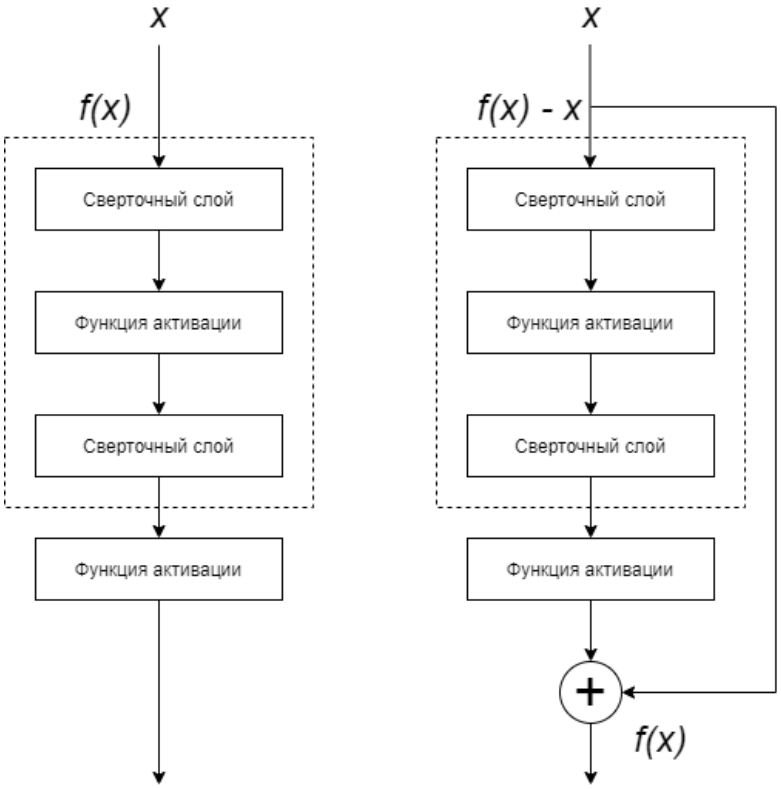

Рисунок 2.9 – Слева обычный блок, справа остаточный блок

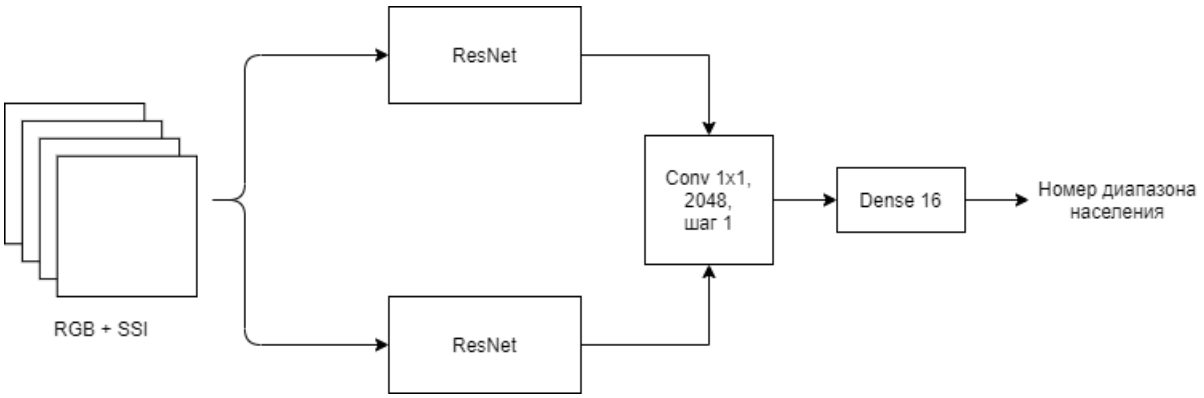

Рисунок 2.10 – Предложенная архитектура прогнозирования диапазона численности населения на четырехканальном изображение

2.5. Улучшение прогноза численности населения

Для улучшения результатов работы системы предлагается использовать набор регрессионных моделей. Использования числовых диапазонов, которые возвращает нейронная сеть прогнозирования диапазона численности населения, в качестве ограничивающих значений, делает задачу получения числа населения такими моделями легче. Схема этой системы показана на рисунке 2.11.

Для их тренировки и последующей работы используются данные, которые получаются после прохода изображений через нейронную сеть, предложенную в предыдущем параграфе, до объединяющего сверточного слоя. Эти данные представляют собой вектор чисел длинной 2048.

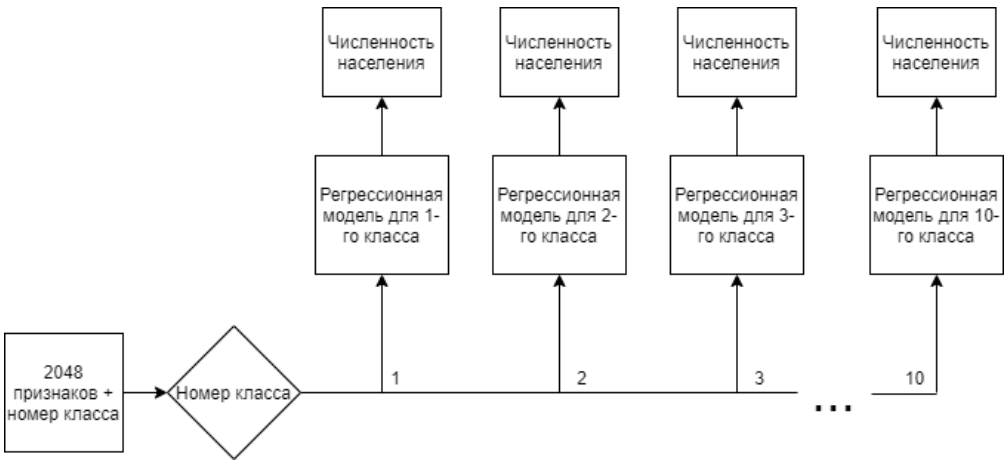

Рисунок 2.11 – Схема применения регрессоров для улучшения результатов.

Глава 3. Реализация и результаты

3.1. Реализация модифицированной сети U-Net++

Над оригинальной архитектурой нейронной сети U-Net++ был проделан ряд изменений целью, которых является адаптация сети для работы со спутниковыми изображениями. Всего было обучено три варианта этой нейронной сети: A, B и C.

Первые три модификации использовались всеми вариантами нейронной сети. Первая модификация заменила функцию активации для всех сверточных слоёв с ReLU (2.8) на Leaky ReLU (3.1) [34]. Эта функция активации почти всегда показывает лучшие результаты по сравнению с функцией ReLU [34]. На рисунке 3.1 показаны графики этих функций.

$$f(x) = \begin{cases} 0.1x, & x < 0 \\ x, & x \geq 0 \end{cases}, \tag{3.1}$$

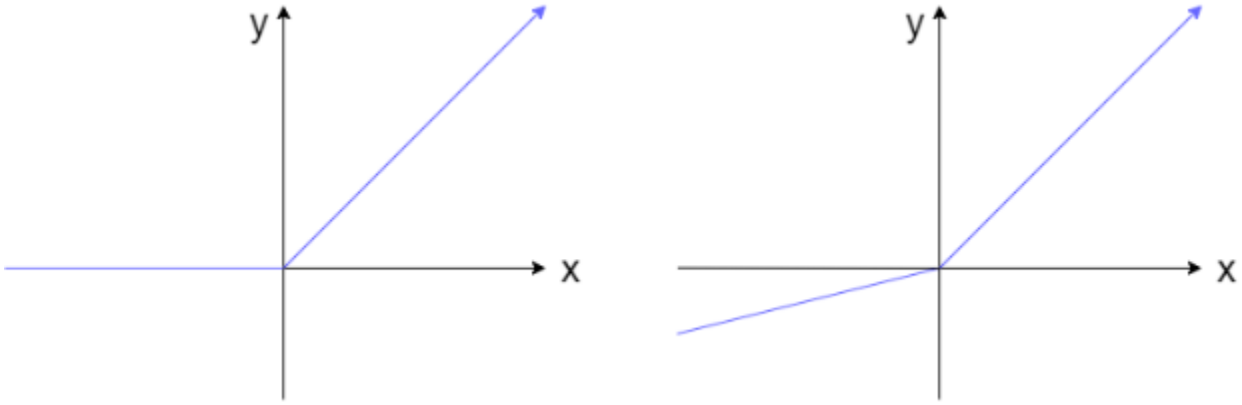

Рисунок 3.1 – Слева график функции активации ReLU, справа график функции активации Leaky ReLU

Вторая модификация является изменением метода увеличения размерности данных внутри сети. Вместо, алгоритма ближайшего соседа во время прямого прохода нейронной сети, использовались транспонированные сверточные слои. Эти слои являются обычными сверточными слоями, но для увеличения размерности тензора проводится ряд дополнительных вычислений. Использование таких

слоев увеличивает количество параметров нейронных сетей, что должно улучшить качество их работы.

Процесс работы транспонированных сверточных слоев можно описать несколькими шагами (3.2-3.4). Первым шагом является вычисление значений z и p', а вторым увеличение размерности данных. Значение z устанавливает количество нулей, которые будут вставлены между каждыми рядами и столбами входных тензоров. Значение p' устанавливает размер отступов, которые будут добавлены к данным. Последним шагом является проход увеличенных данных через обычный сверточный слой с шагом $s' = 1$.

$$z = s - 1, \tag{3.2}$$

$$p' = k - p - 1, \tag{3.3}$$

$$o = (i - 1) \times s + k - 2p, \tag{3.4}$$

где $s -$ шаг,

k – размер окна свертки,

$i -$ размер начальных данных,

$o -$ размер увеличенных данных,

$s -$ размер отступа для сверточного слоя.

На рисунке 3.2 графически продемонстрирован полный процесс прохода тензоров через транспонированный сверточный слой.

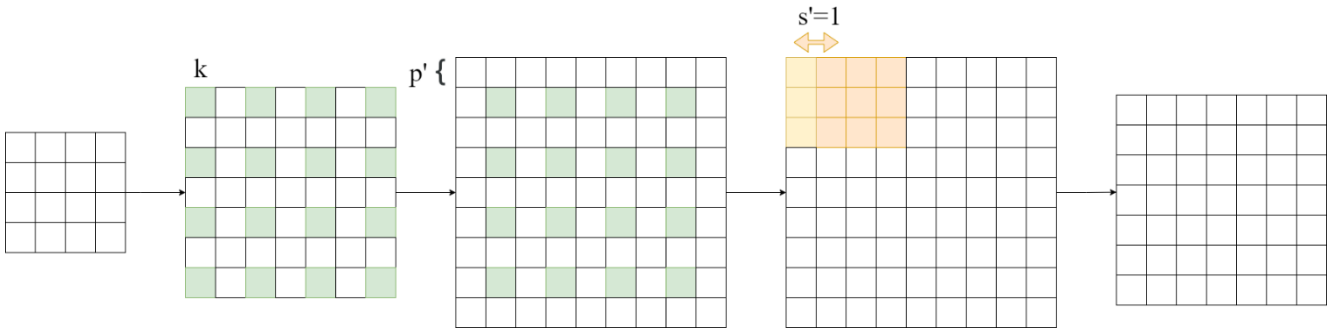

Рисунок 3.2 – Процесс прохода матрицы 4×4 через транспонированный сверточный слой с шагом $s = 1$, отступом $p = 1$ и окном свертки размером $k = 3$

Последним изменением является увеличение количества сверточных слоев до трех в каждом блоке пропускных путей и декодера, что увеличивает сложность нейронной сети.

Вариант A использовал функцию потери, которая была предложена в оригинальной статье (2.10). Варианты B и C использовали комбинированную функцию потери (3.5), предложенную в статье [35]:

$$L = L_{mF} + (1 - \lambda)L_{mFD}, \tag{3.5}$$

где λ – балансирующий коэффициент,

L_{mF} – модифицированная фокальная функция потери,

L_{mFD} – фокальная версия функции потери Дайса.

Идея позади комбинирования функций потерь является получение такой функции, которая имеет положительные свойства всех функций, из которых она состоит.

Главное свойство модифицированной фокальной функции потери является ее хорошая способность работать с сильно несбалансированными классами входных данных. Она достигает этого с помощью уменьшения влияния корректно спрогнозированных значений на финальный результат значения потери. По сколько рассматриваемый набор данных демонстрирует несбалансированность, а именно площадь класса незастроенной территории больше половины общей площади. На таблице 3.1 показано распределение площадей покровов земли разных классов. Рассмотрим саму модифицированную функцию потери (3.6-3.8):

$$\mathrm{L}_{mF} = -\alpha(1 - p_t)^{\gamma} L_{mCE}, \tag{3.6}$$

$$p_t = \begin{cases} p, & \text{если } t = 1 \\ 1 - p, & \text{если } t = 0 \end{cases}, \tag{3.7}$$

$$L_{mCE} = -\frac{1}{N}\sum_{i=1}^{N} \beta(t_i - \log(p_i)) + (1 - \beta)[(1 - t_i)\ln(1 - p_i)], \tag{3.8}$$

где L_{mCE} – модифицированная функция потери перекрестной энтропии,

α – параметр, контролирующий веса классов при подсчете значения потери,

$\gamma -$ параметр, контролирующий влияние корректно спрогнозированных значений, $\beta -$ параметр, контролирующий веса неправильно спрогнозированных значений, p и $t -$ спрогнозированные и настоящие значения классов, соответственно.

Таблица 3.1 – Площади, которые занимают разные классы покровов земли

Номер класса	1	2	3	4	5	6
Площадь, %-ов	7.66	11.972	13.392	9.5	3.717	53.752

Функция потери Дайса (3.9–3.10) одна из самых популярных функций для задачи семантической сегментации изображений. Спрогнозированные и настоящие классы можно рассматривать как два набора чисел. В процессе обучения функция потери Дайса поощряет пересечения между этими наборами.

$$mDSC = \frac{\sum_{i=1}^{N} p_{0i} t_{0i}}{\sum_{i=1}^{N} p_{0i} t_{0i} + \delta \sum_{i=1}^{N} p_{0i} g_{1i} + (1-\delta) \sum_{i=1}^{N} p_{1i} g_{0i}}, \quad (3.9)$$

$$L_{mFD} = \sum_{c=1}^{C} (1 - mD)^{\frac{1}{\gamma}}, \quad (3.10)$$

где p_{0i} и $p_{1i} -$ вероятности пикселей принадлежать к корректным и некорректным классам соответственно,

$g_{0i} -$ равно 1 если пикселей принадлежит корректному классу и 0 если принадлежит некорректному,

$\mathrm{g}_{1i} -$ обратно g_{0i},

$\gamma -$ фокальный параметр эквивалентный параметру γ в фокальной функции потери.

В итоге результирующая функция потери (3.5) способна гладко сходится для задач сегментации и работать с сильно несбалансированными данными.

Разница между вариантами B и C является размер сверточного окна в первом слое каждого блока пропускных путей. Вариант B имеет размер $k = 3$, тогда как вариант C имеет размер $k = 5$. Идея позади увеличения размера окна является уменьшение чувствительности нейронной сети к более мелким деталям таким, как дороги, но улучшение способности локализировать более крупные скопления одноклассовых районов на изображениях.

Для обучения и тестирования всех вариантов нейронной сети U-Net++ использовалось 163,840 и 32,768 изображений соответственно, что больше начального набора данных. Новые изображения были получены с помощью операций поворотов вокруг осей X и Y. С целью балансирования классов были взяты только 50% изображений с населением в первых двух диапазонах и 70% изображений с населением в первых 5-ом и 6-ом диапазонах. На рисунке 3.3 продемонстрирована гистограмма финального распределения изображений по диапазонам численности населения.

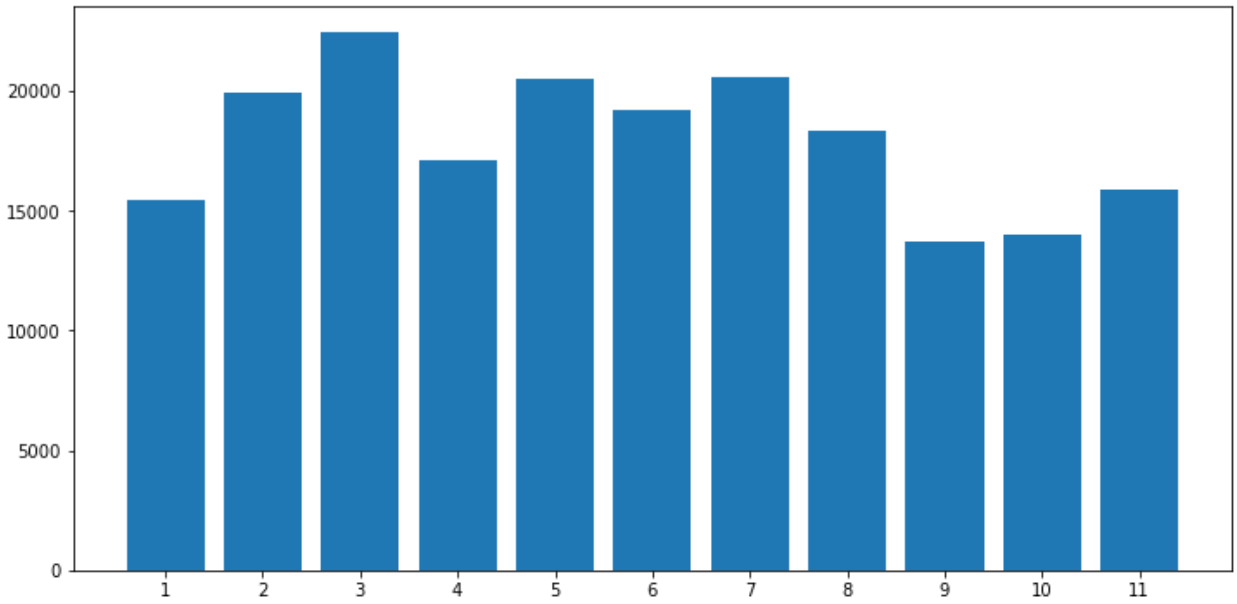

Рисунок 3.3 – Гистограмма классов после процесса расширения данных

В качестве энкодера во всех вариантах использовалась пре-тренированная сверточная нейронная сеть VGG-19. Это позволяет энкодеру уже до начала тренировки иметь способность различать общие детали такие, как линии, углы и тек-

стуру изображений, что ускоряет процесс тренировки и увеличивает общее качество работы нейронной сети.

Все варианты тренировались с коэффициентом скорости обучения 0.0001, размером пакета 16 и алгоритмом оптимизации Adam. Для регуляризации нейронных сетей использовался приём ренормализации пакетов [36], dropout слои [37] и сглаживание ярлыков [38]. Варианты B и C тренировалась 5 полных эпох, а тренировка варианта A была остановлена после двух неполных эпох обучения. На таблице 3.2 показаны результаты тренировок разных вариантов архитектуры нейронной сети U-Net++. На рисунках 3.4–3.9 показаны графики по пиксельной точности и метрики mean IoU.

Таблица 3.2 – Результаты тренировки сети U-Net++

Вариант	Максимальная достигнутая по пиксельной точности, %	Максимальное достигнутое значение Mean IoU
A	71.45	0.4381
B	75.31	0.4998
C	75.61	0.5022

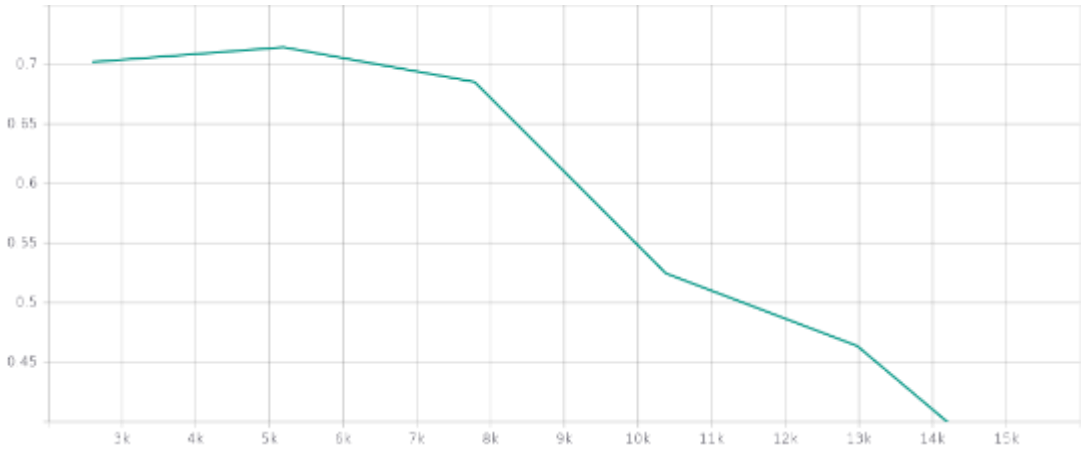

Рисунок 3.4 – График точности версии A нейронной сети U-Net++

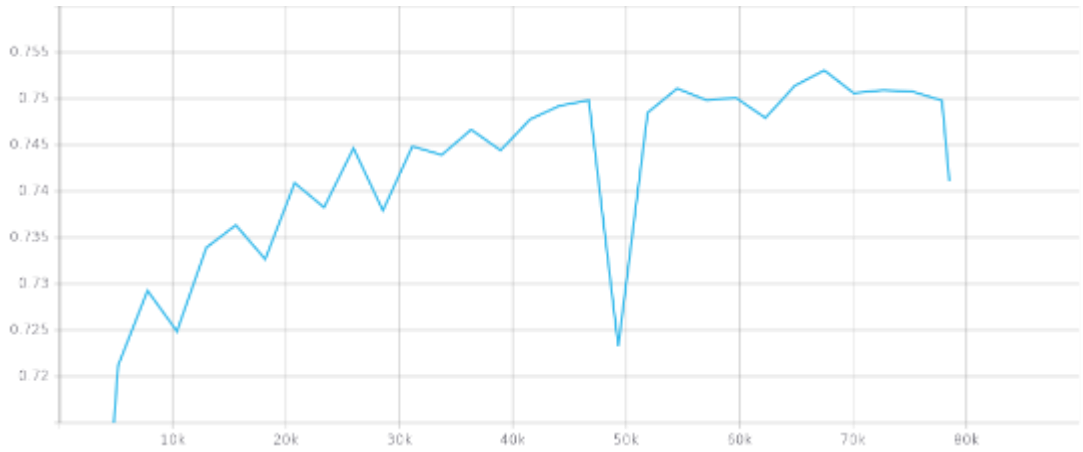

Рисунок 3.5 – График точности версии B нейронной сети U-Net++

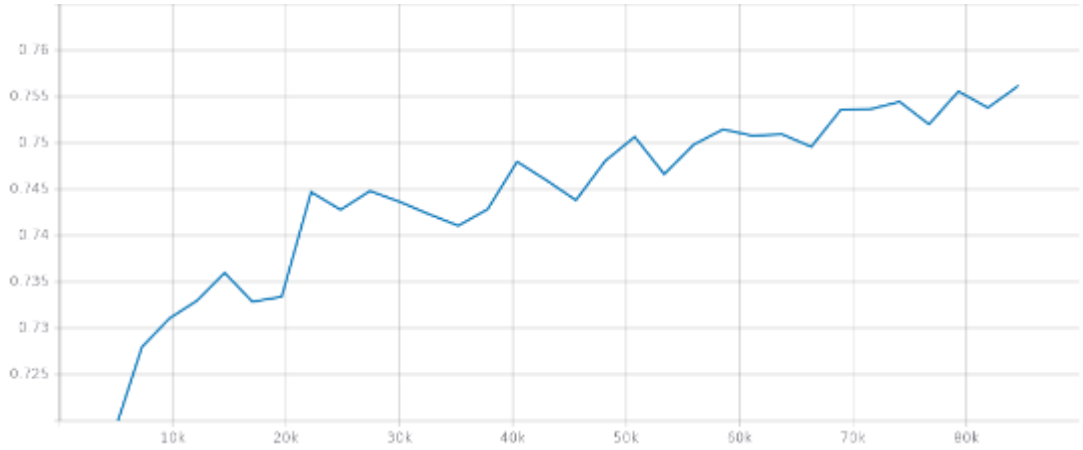

Рисунок 3.6 – График точности версии C нейронной сети U-Net++

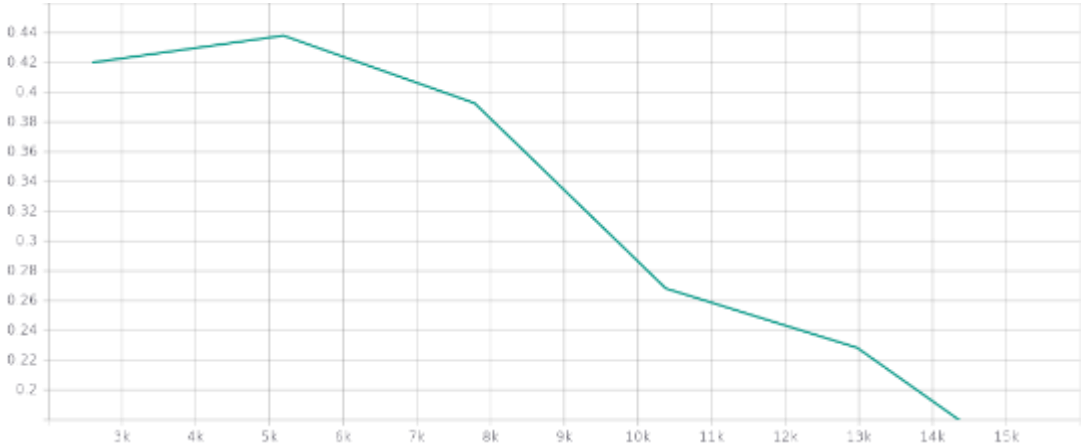

Рисунок 3.7 – График значения метрики mean IoU версии A нейронной сети U-Net++

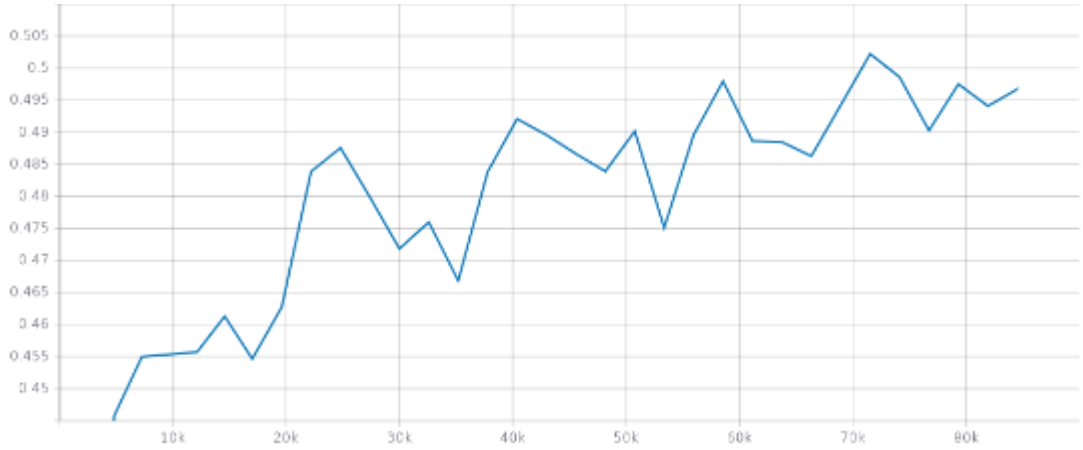

Рисунок 3.8 – График значения метрики mean IoU версии A нейронной сети U-Net++

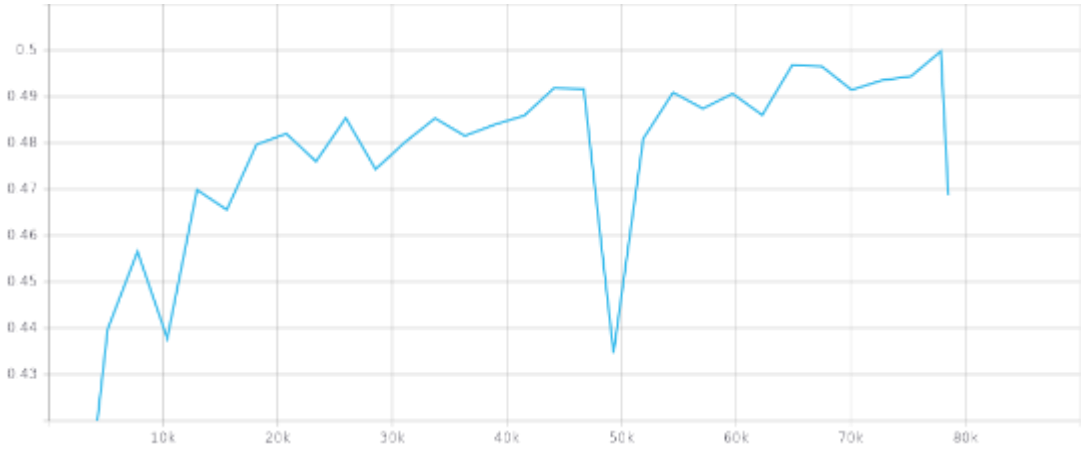

Рисунок 3.9 – График значения метрики mean IoU версии C нейронной сети U-Net++

Как видно по графикам и таблице результатов тренировки сеть U-Net++ с начальной функцией потери не может справится с крайней несбалансированностью данных, что делает тренировку данного варианта невозможной. Вариант C показывает более стабильный процесс тренировки и небольшое увеличение качества работы сети по сравнению с вариантом B.

На рисунке 3.10 показан пример работы сети U-Net++. Больше примеров можно найти в приложении A.

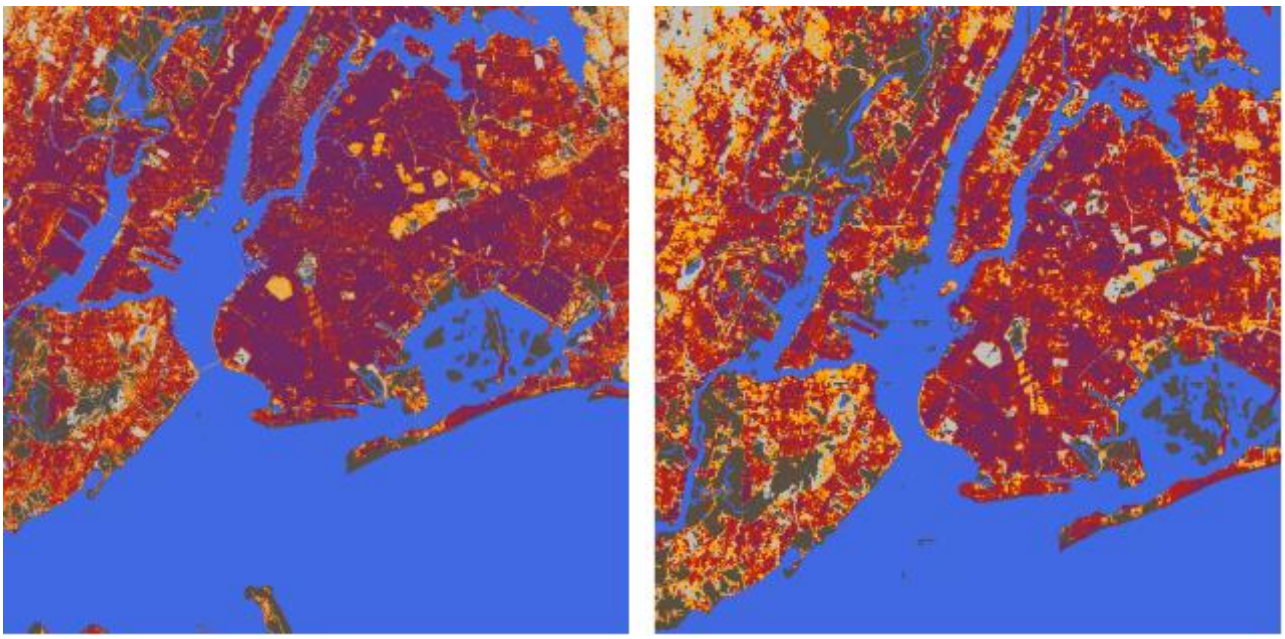

Рисунок 3.10 – Полученный (слева) и настоящий (справа) покров земли Нью-Йорка.

3.2. Реализация сети прогнозирования диапазона численности населения

Всего было обучено 4 нейронных сетей для прогнозированная диапазона численности населения. Первая нейронная сеть обучалась только на RGB изображения. Для всех остальных входные RGB изображения сначала пропускались через нейронную сеть U-Net++ для получения карты их земного покрова (SSI), а затем эти карты использовались в качестве четвертого канала для входных изображений для нейронных сетей прогнозирования диапазона численности населения.

Первая и вторая нейронные сети использовали архитектуру ResNet-101. Архитектура этой сети показана на таблице 3.3.

Третья и четвертая сети использовали архитектуру, предложенную в главе 2.4. В качестве нейронных сетей на ветвях использовалась та же архитектура ResNet-101.

Таблица 3.3 – Архитектура сети ResNet-101

Название блока	Размер данных на выходе	Слои
Conv1	64×64	$7 \times 7, \text{фильтры} = 64, \text{шаг} = 2$
Conv2	32×32	3×3 max pool, шаг = 2 $\begin{bmatrix} 1 \times 1, \text{фильтры} = 64 \\ 3 \times 3, \text{фильтры} = 64 \\ 1 \times 1, \text{фильтры} = 256 \end{bmatrix} \times 3$
Conv3	16×16	$\begin{bmatrix} 1 \times 1, \text{фильтры} = 128 \\ 3 \times 3, \text{фильтры} = 128 \\ 1 \times 1, \text{фильтры} = 512 \end{bmatrix} \times 4$
Conv4	8×8	$\begin{bmatrix} 1 \times 1, \text{фильтры} = 256 \\ 3 \times 3, \text{фильтры} = 256 \\ 1 \times 1, \text{фильтры} = 1024 \end{bmatrix} \times 23$
Conv5	4×4	$\begin{bmatrix} 1 \times 1, \text{фильтры} = 512 \\ 3 \times 3, \text{фильтры} = 512 \\ 1 \times 1, \text{фильтры} = 2048 \end{bmatrix} \times 3$

Первые три нейронные сети использовали функцию потери перекрестной энтропии (3.11):

$$L = \sum_{i=1}^{C} t_i \log p_i \,, \tag{3.11}$$

где $C -$ количество классов,

$t_c -$ бинарный индикатор принадлежности изображения к классу i,

$p_i -$ полученная вероятность изображения принадлежать к классу i.

Четвертая нейронная сеть использовала функцию потери, основанную на метрике Earth Mover's Distance (EMD) [39] (3.12):

$$L = \sum_{i=1}^{C} \left(CDF_i(p) - CDF_i(t)\right)^2 , \tag{3.12}$$

где $CDF -$ кумулятивная функция распределения,

$C -$ количество классов,

$p -$ полученная вероятность изображения принадлежать к классу i,

$t -$ бинарный индикатор принадлежности изображения к классу i.

Эта функция позволяет учитывать близость спрогнозированного класса изображения к его настоящему значению в упорядоченном списке классов, то есть, чем ближе полученный нейронной сетью класс находится к настоящему значению, тем меньше он будет влиять на вычисляемое значение потери. В задаче прогноза населения это важно, поскольку ошибка между соседними диапазонами сделает только небольшую числовое изменение при подсчёте населения, тогда как ошибка между дальними диапазонами сильно уменьшит точность подсчета.

Все сети тренировались и тестировались на тех же изображениях, которые были использованы для сети сегментации Длинна тренировки варьировалась для каждой модели, но все достигли свою максимальную точность на 9-ом или 10-ом проходе по данным. Использовался коэффициент скорости обучения 0.0001, размер пакета 64 и алгоритм оптимизации Adam. Результаты тренировки нейронных сетей приведены на таблице 3.4, графики точности и матрицы ошибок продемонстрированы на рисунках 3.11–3.14, пример спутникового снимка с наложенной на него тепловой картой классов диапазонов численности населения показан на рисунке 3.15.

На таблице 3.5 сравнены настоящие и полученные численности населения городов, не участвующих в процессе тренировки. Для получения числа каждому классу было задано число равное среднему значению его диапазона.

Как можно заметить на графиках и матрицах ошибок третья нейронная сеть натренированная на дополненных картой покрова земли изображениях дала лучше результат, чем одиночные нейронная сеть, что эмпирически доказывает, что использование семантически сегментированных спутниковых изображений в качестве дополнительного канала может увеличить качество работы нейронной сети прогнозирования диапазона численности населения.

Функция потери, основанная на Earth Mover’s Distance, не показала улучшения в качестве работы нейронной сети.

Таблица 3.4 – Результаты тренировок сетей прогнозирования населения

Модель	Resnet-101	Resnet-101	Параллельная архитектура	Параллельная архитектура с функцией потери EMD
Данные	RGB	RGB + SSI	RGB + SSI	RGB + SSI
Максимальная точность	50.88 %	50.28 %	52.95 %	47.81%
Номер эпохи	10	9	9	9

Талица 3.5 – Спрогнозированные численности населения городов, не участвующих в тренировке нейронной сети

Город	Настоящая численность населения	Полученная численность
Талса	397,087	494,985
Солт-Лейк-Сити	526,464	616,277
Сан-Антонио	738,531	769,270
Юджин	164,509	192,492
Денвер	1,029,655	1,024,535

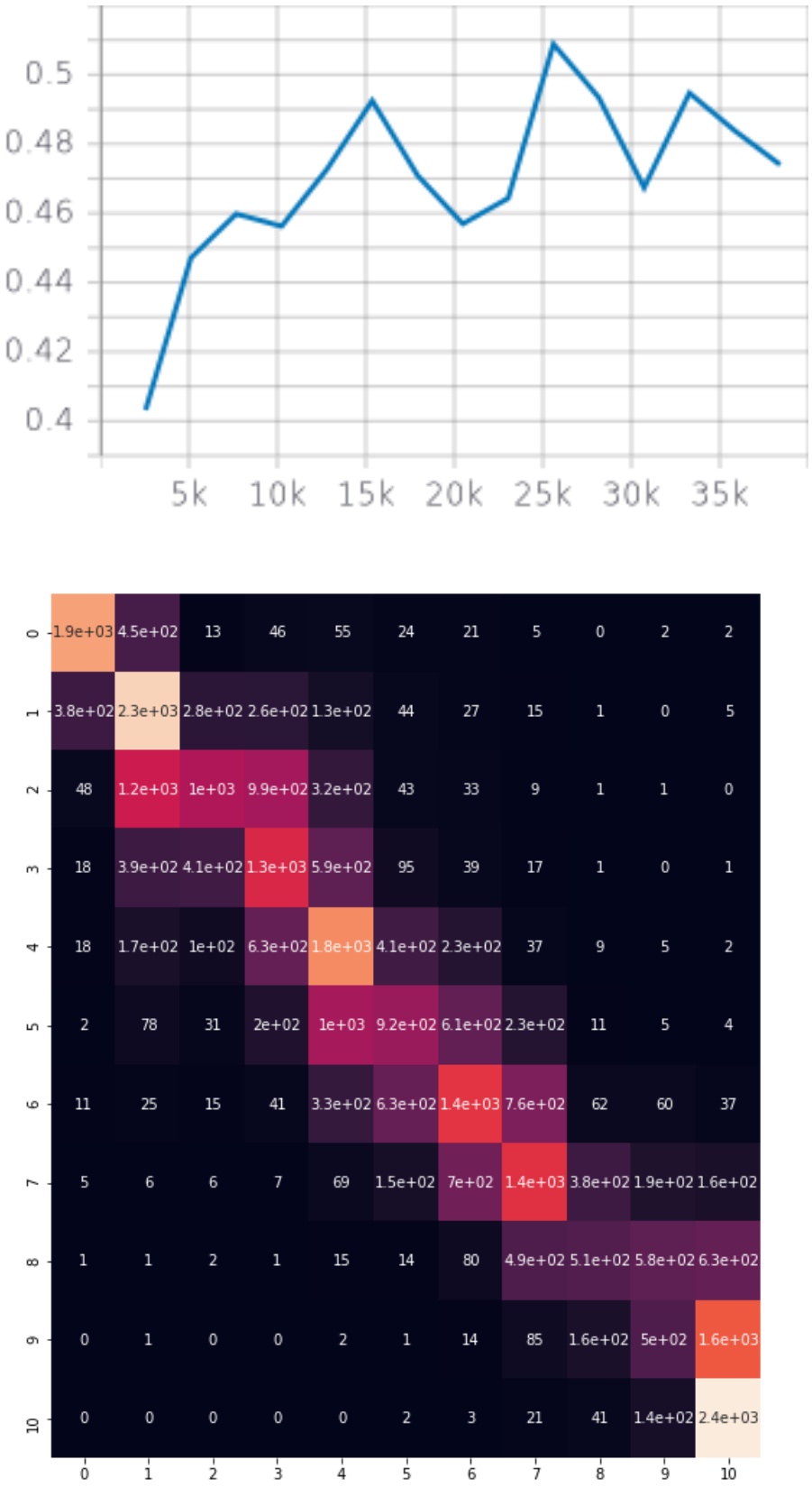

Рисунок 3.11 – График точности и матрица ошибок первой нейронной сети прогнозирования населения

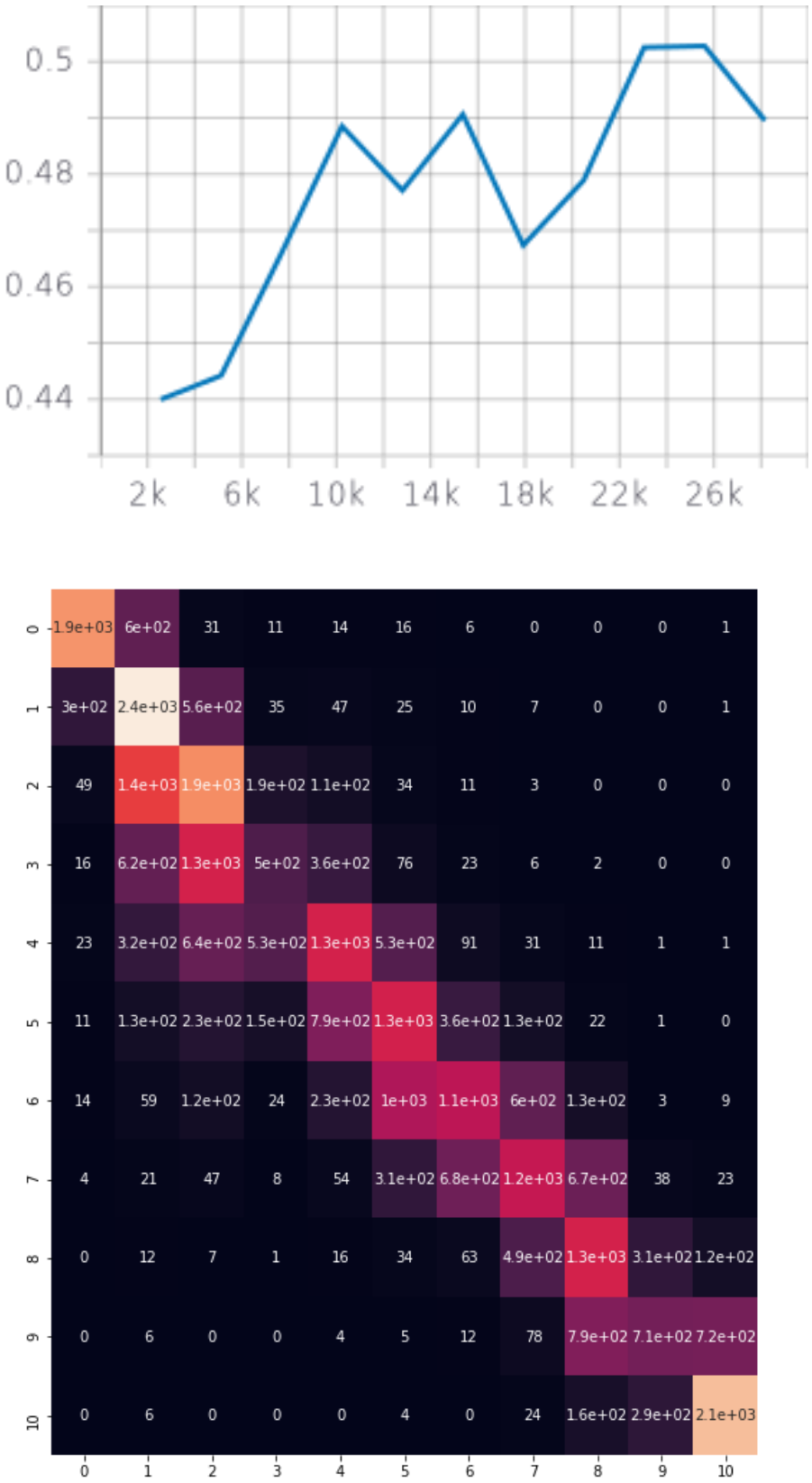

Рисунок 3.12 – График точности и матрица ошибок второй нейронной сети прогнозирования населения

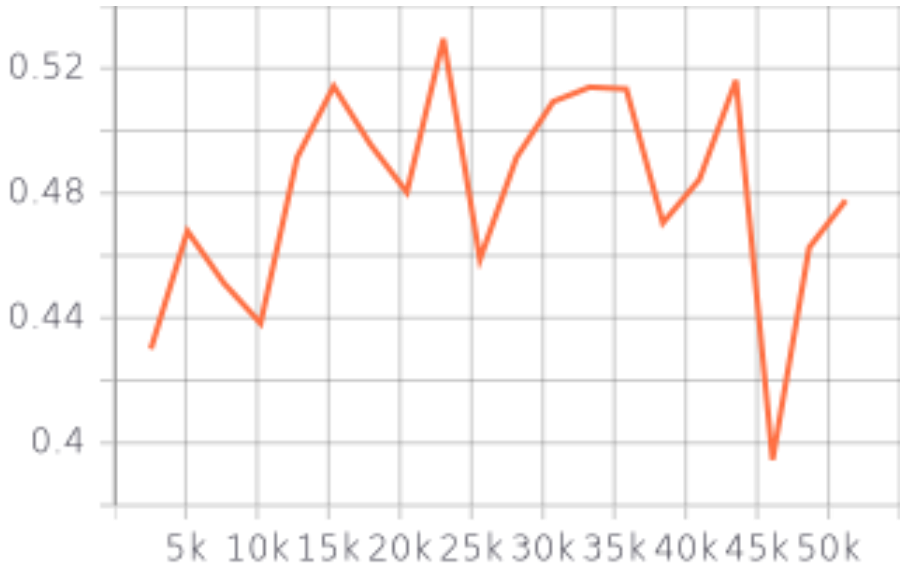

Рисунок 3.13 – График точности и матрица ошибок третьей нейронной сети прогнозирования населения

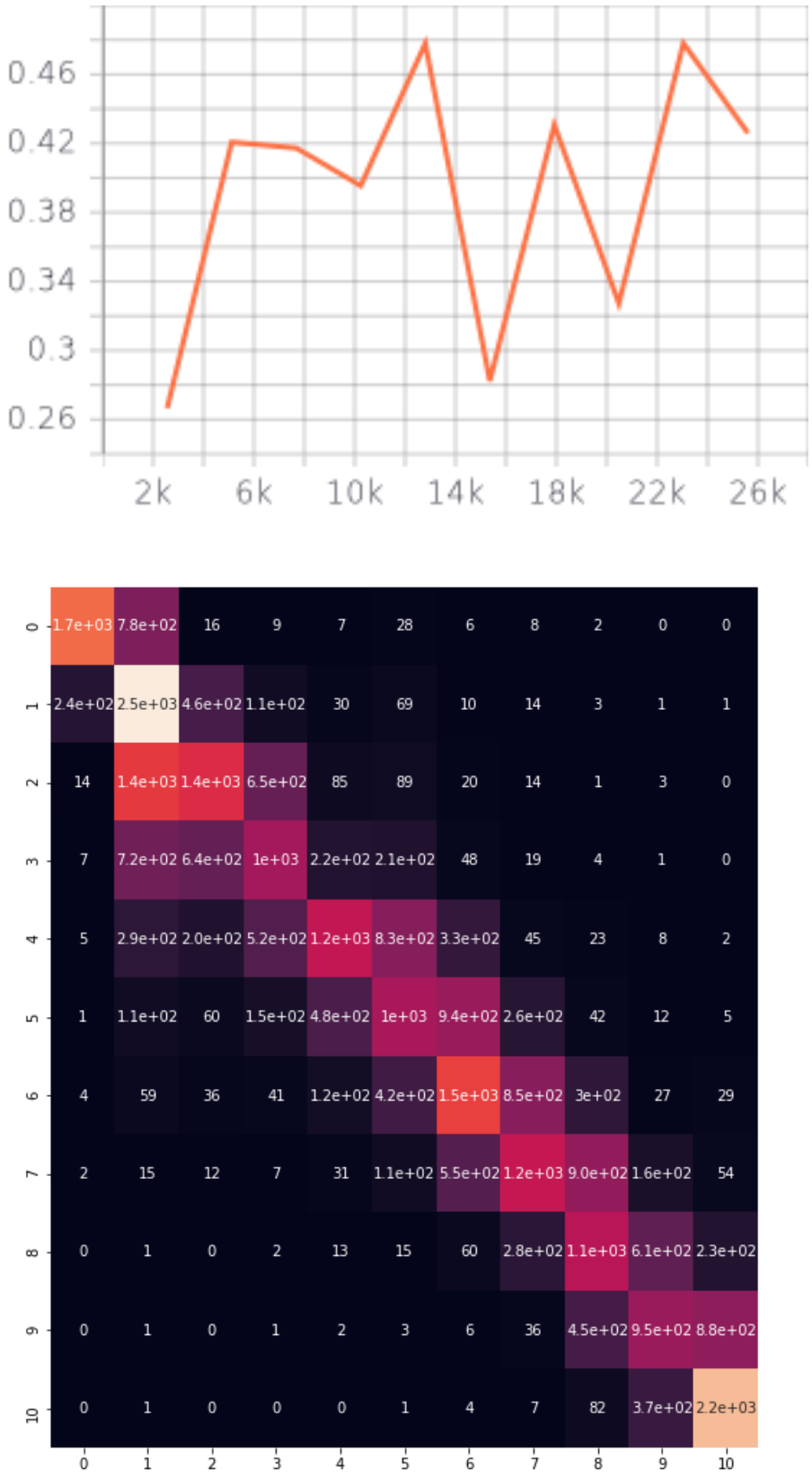

Рисунок 3.14 – График точности и матрица ошибок четвертой нейронной сети прогнозирования населения

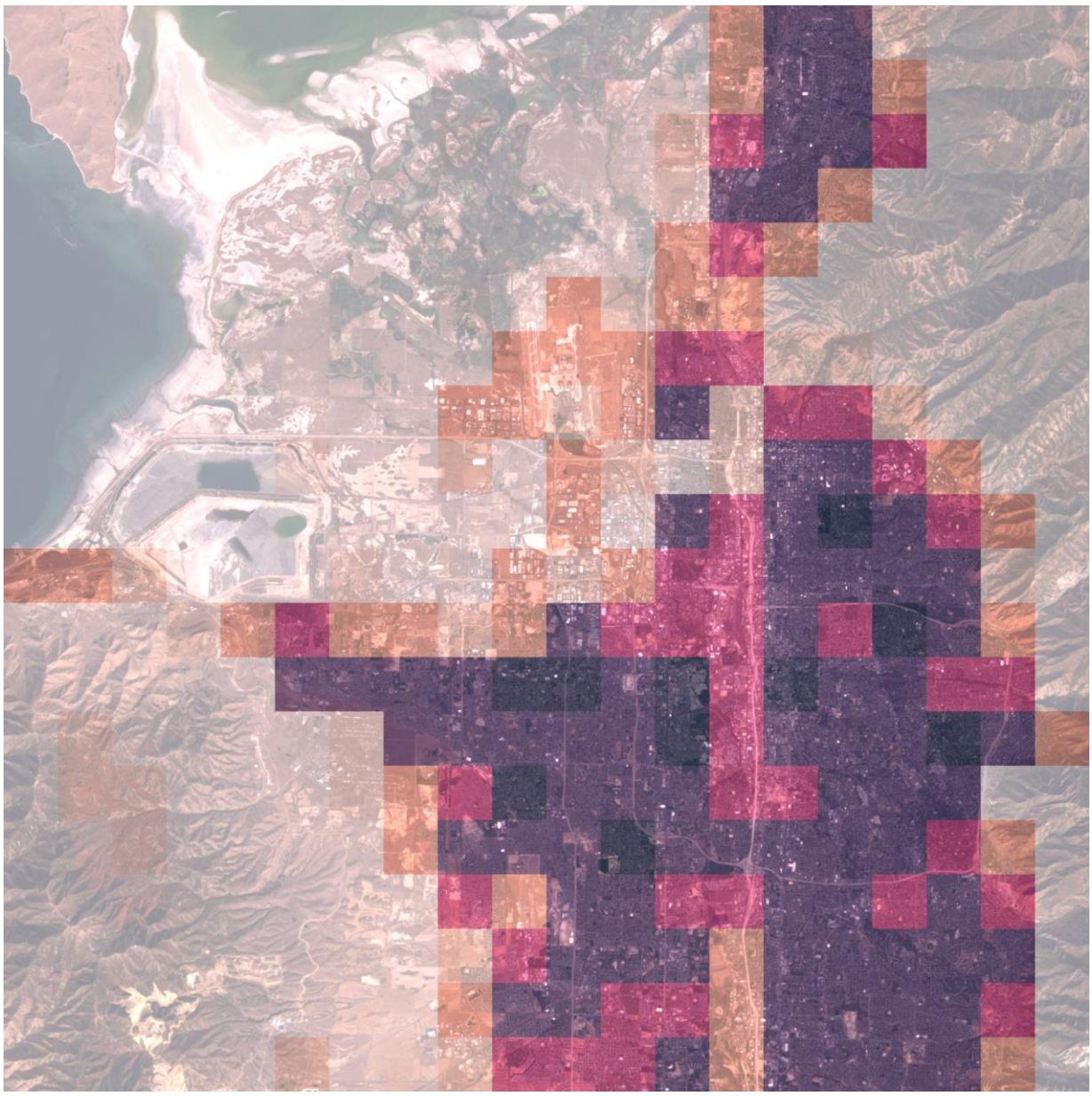

Рисунок 3.15 – Пример спутникового снимка с тепловой картой полученного населения

Больше примеров работы этой модели показано в приложении.

3.3. Реализация регрессионных моделей

В качестве регрессионной модели был выбран метод опорных векторов для регрессии [40].

Метод опорных векторов для регрессии использует набор данных x и вектор измеренных значение y, находит линейную функцию (3.13-3.14) с минимальным значением нормы (β', β) и выполняющие условие (3.15):

$$f(x) = x'\beta + b, \tag{3.13}$$

$$J(\beta) = \frac{1}{2}\beta'\beta \ \rightarrow \ min, \tag{3.14}$$

$$\forall n: |y_n - (x_n'\beta + b)| \leq \varepsilon. \tag{3.15}$$

На таблице 3.6 показаны результаты тренировок этих моделей, а на таблице 3.7 полученные моделями численности населения для городов из таблицы 3.5.

Таблица 3.6 – Результаты тренировки регрессионных моделей.

Номер модели	1	2	3	4	5
Метрика MAE	2.726	5.648	13.633	23.563	69.164
Номер модели	6	7	8	9	10
Метрика MAE	72.267	290.306	488.62	483.193	4147.53

Таблица 3.7 – Результаты тренировки регрессионных моделей

Город	Настоящая численность населения	Полученная численность до регрессионных моделей	Полученная численность после регрессионных моделей
Талса	397,087	494,985	463,715
Солт-Лейк-Сити	526,464	616,277	613,409
Сан-Антонио	738,531	769,270	750,449
Юджин	164,509	192,492	182,691
Денвер	1,029,655	1,024,535	1,056,849

Как видно на таблице 3.7 результат работы системы улучшился для четырёх из пяти городов.

Заключение

В ходе проделанной работы были получены следующие результаты:

- Построена система прогнозирования численности населения
- Модифицирована архитектура нейронная сеть для семантической сегментации спутниковых изображений.
- Разработана архитектура нейронной сети для классификации диапазонов численности населения.
- Натренированы регрессионные модели численности населения.
- Разработан алгоритм подготовки данных и натренированы все участвующие нейронные сети в системе.

Следующие меры могут улучшить качество работы системы:

- Дальнейшее обучение модели сегментации должно улучшить ее работу.
- Использование ансамблевых методов для регрессии.
- Увеличение выборки данных с 19 снимков.
- Использование методов регуляризации и более сложных функций активаций может привести к росту точности нейронных сетей.
- Увеличение количества диапазонов населения и их более тщательный подбор.
- Использование более глубокого декодера в сети сегментации.

Список литературы.

1. Landsat 7: Description of Spectral Bands. https://landsat.usgs.gov.
2. G. Palubinskas. Model-based image adjustment for a successful pansharpening / ArXiv, abs/2103.03062 (2021).
3. NLCD 2001 Land Cover (CONUS). https://www.mrlc.gov/data/nlcd-2001-land-cover-conus.
4. U.S. Census Grids. https://sedac.ciesin.columbia.edu/data/set/usgrid-summary-file1-2010/data-download.
5. T. Ojala. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns / IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 7, pp. 971–987, July 2002.
6. D. Lowe. Distinctive image features from scale-invariant key points / International Journal of Computer Vision, vol. 60, no. 2, pp. 91–110, 2004.
7. N. Dalal. Histograms of oriented gradients for human detection / IEEE International Conference on Computer Vision and Pattern Recognition, 2005, pp. 886-893.
8. Y. Yang. Bag-of-visual-words and spatial extensions for land-use classification. / International Conference on Advances in Geographic Information Systems, 2010, pp. 270–279.
9. Y. Yang. Spatial pyramid co-occurrence for image classification / IEEE International Conference on Computer Vision, 2011, pp. 1465–1472.
10. Y. Zhang. High-resolution remote-sensing image classification via an approximate earth movers distance-based bag-of-features model / IEEE Geoscience and Remote Sensing Letters, vol. 10, no. 5, pp. 1055–1059, September 2013.
11. A. Barsi. Artificial neural networks for the detection of road junctions in aerial images / Int. Arch. Of Photogrammetry Remote Sensing and Spatial Inf. Sciences, vol. 34, pp. 113–118, 2003.
12. O. Firat. Representation learning for contextual object and region detection in remote sensing / ICPR, pp. 3708–3713, 2014.

13. C. Hung. Feature learning based approach for weed classification using high resolution aerial images from a digital camera mounted on a UAV / RemoteSensing, 6(12):12037–12054, 2014.
14. C. Robinson. A Deep Learning Approach for Population Estimation from Satellite Imagery / ArXiv, abs/1708.09086 (2017).
15. M. Castelluccio. Land Use Classification in Remote Sensing Images by Convolutional Neural Networks / ArXiv, abs/1508.00092 (2015).
16. M. Xie. Transfer learning from deep features for remote sensing and poverty mapping / ArXiv, abs/1510.00098 (2015).
17. G. Scarpa. Target-adaptive CNN-based pansharpening / ArXiv, abs/1709.06054 (2017).
18. A. Albert. Modeling urbanization patterns with generative adversarial networks / ArXiv, abs/1801.02710 (2015).
19. S. Ganguli. GeoGAN: A Conditional GAN with Reconstruction and Style Loss to Generate Standard Layer of Maps from Satellite Images / ArXiv, abs/1902.05611 (2019).
20. S. Basu. DeepSat – A Learning framework for Satellite Imagery / ArXiv, abs/1509.03602 (2015).
21. X. Liu. Recent progress in semantic image segmentation / ArXiv, abs/1809.10198 (2018).
22. X. Liu. Importance-Aware Semantic Segmentation in Self-Driving with Discrete Wasserstein Training / ArXiv, abs/2010.12440 (2020).
23. S. Asgari. Deep Semantic Segmentation of Natural and Medical Images: A Review / ArXiv, abs/1910.07655 (2019).
24. J. Long. Fully Convolutional Networks for Semantic Segmentation / ArXiv, abs/1411.4038 (2014).
25. O. Ronneberger. U-Net: Convolutional Networks for Biomedical Image Segmentation / ArXiv, abs/1505.04597 (2015).
26. O. Oktay. Attention U-Net: Learning Where to Look for the Pancreas / ArXiv, abs/1804.03999 (2018).

27. E. Schönfeld. A U-Net Based Discriminator for Generative Adversarial Networks / ArXiv, abs/2002.12655 (2020).
28. Z. Zhou. UNet++: A Nested U-Net Architecture for Medical Image Segmentation / ArXiv, abs/1807.10165 (2018).
29. A. Albert. Using Convolutional Networks and Satellite Imagery to Identify Patterns in Urban Environments at a Large Scale / ArXiv, abs/1704.02965 (2017).
30. V. Khryashchev. Comparison of Different Convolutional Neural Network Architectures for Satellite Image Segmentation / pp. 172-179. 10.23919 / FRUCT.2018. 8588071 (2018).
31. C. Lee. Deeply-Supervised Nets / ArXiv, abs/1409.5185 (2014).
32. G. Huang. Densly Connected Convolutional Networks / ArXiv, abs/1608.06993 (2016).
33. K. He. Deep Residual Learning for Image Recognition / ArXiv, abs/1512.03385 (2015).
34. B. Xu. Empirical Evaluation of Rectified Activations in Convolution Network / ArXiv, abs/1505.00853 (2015).
35. M. Yeung. A Mixed Focal Loss Function for Handling Class Imbalanced Medical Image Segmentation / ArXiv, abs/2102.04525 (2021).
36. S. Ioffe. Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models / ArXiv, abs/1702.03275 (2017).
37. S. Cai. Efficient and Effective Dropout for Deep Convolutional Neural Networks / ArXiv, abs/1904.03392 (2019).
38. C. Zhang. Delving Deep into Label Smoothing / ArXiv, abs/2011.12562 (2020).
39. L. Hou. Squared Earth Mover’s Distance-based Loss for Training Deep Neural Networks / ArXiv, abs/1611.05916 (2016).
40. H. Drucker. Support Vector Regression Machines / Advances in Neural Information Processing Systems 9, 1996, pp. 155-161.

Приложение А. Примеры работы нейронной сети U-Net++.

(справочное)

Пример работ варианта С нейронной сети U-Net++ на снимках городах, которые не использовались в процессе тренировки.

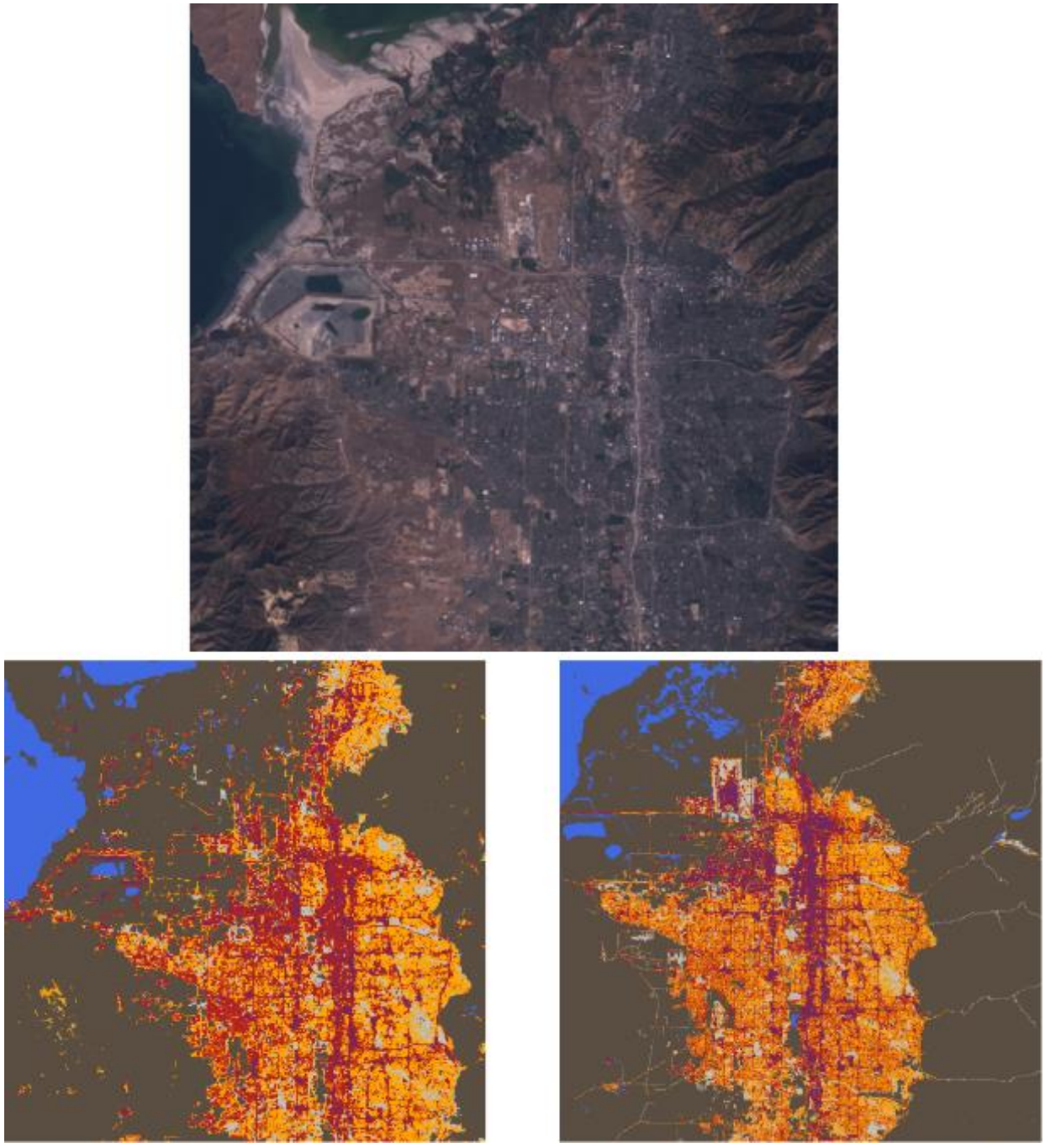

Рисунок А.1. – Результат обработки спутникового снимка Солт-Лейк-Сити. По часовой стрелке: входное изображение, полученный земной покров, настоящей земной покров

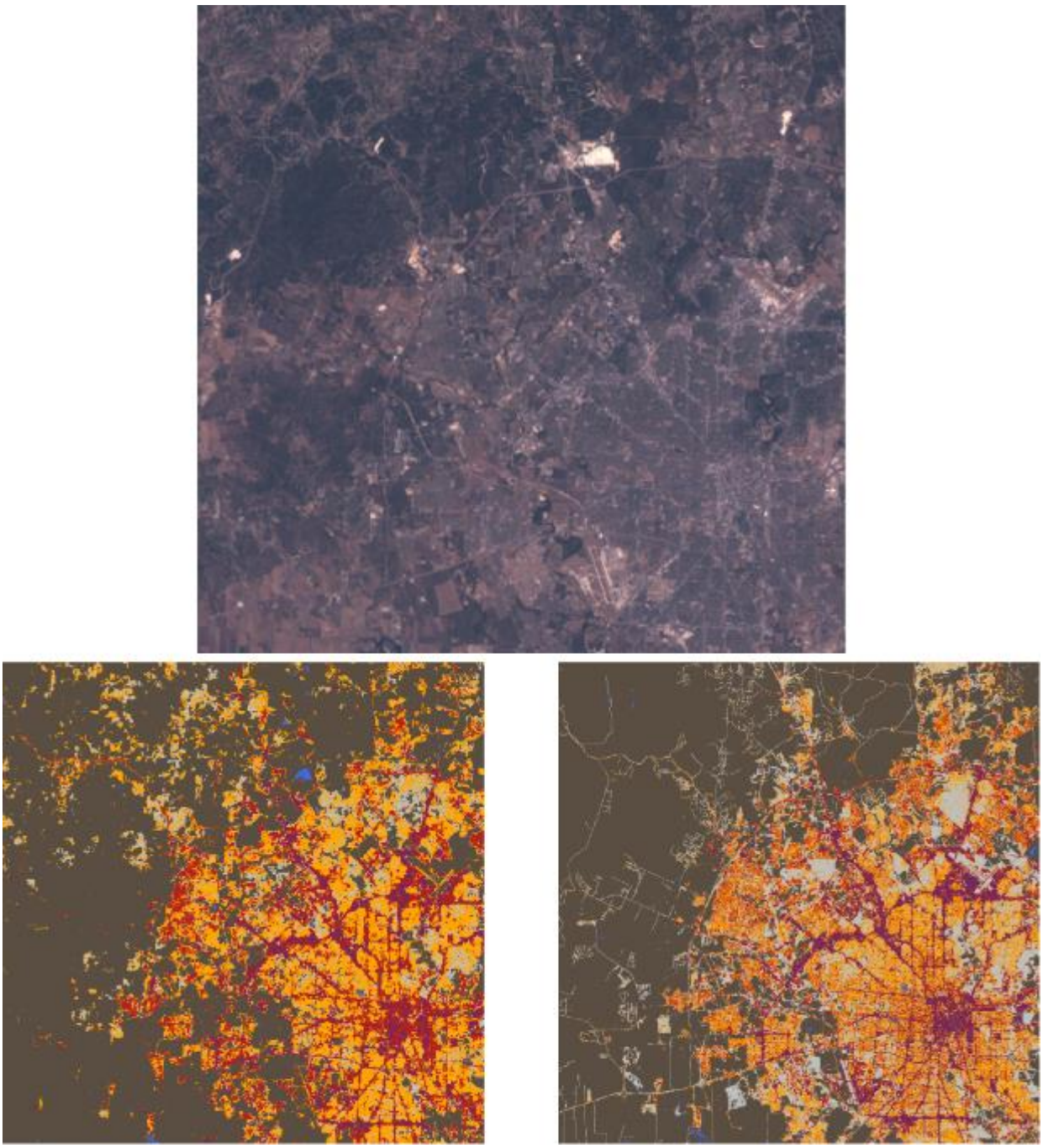

Рисунок А.2. – Результат обработки спутникового снимка Сан-Антонио. По часовой стрелке: Входное изображение, полученный земной покров, настоящей земной покров

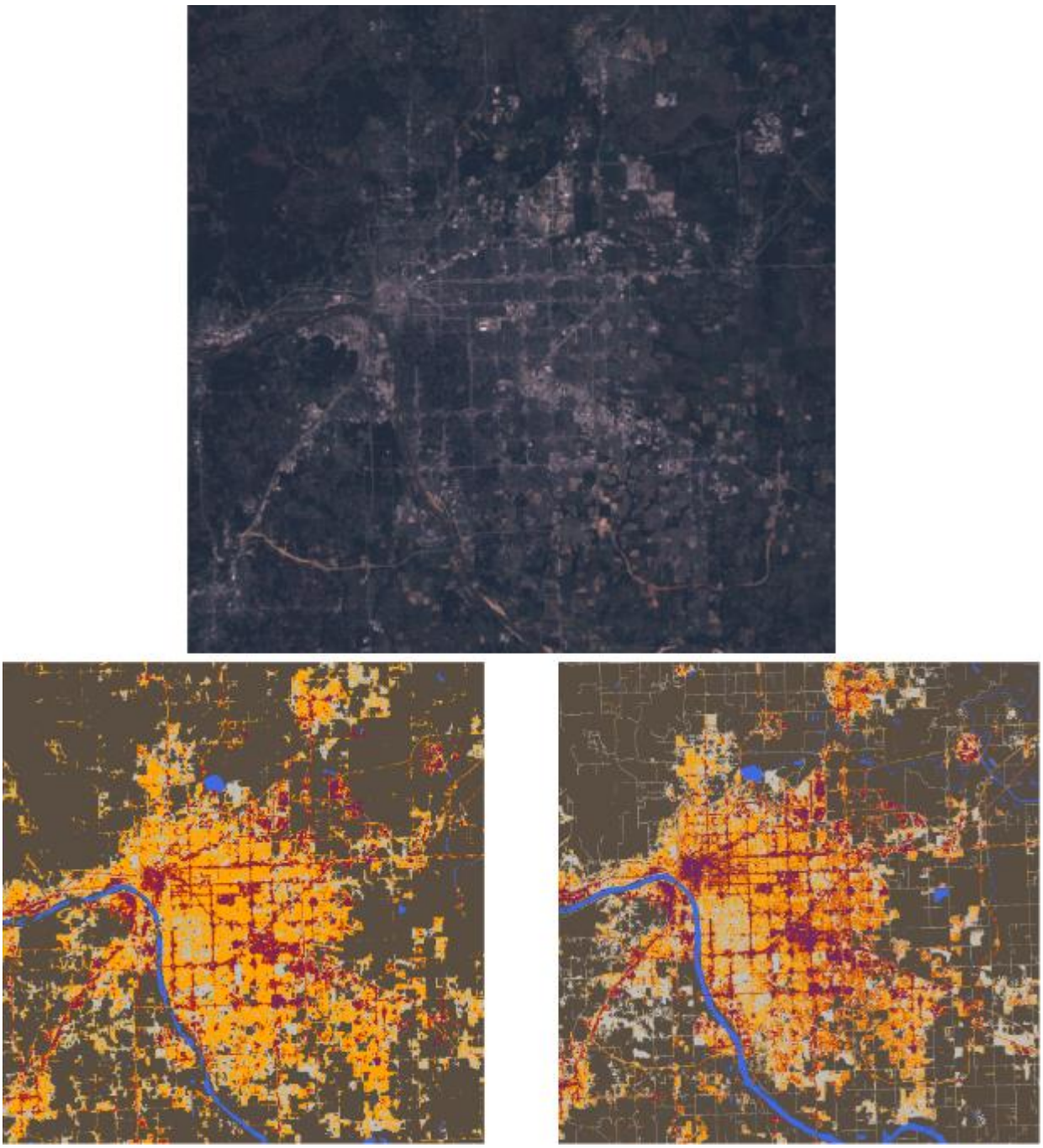

Рисунок А.2. – Результат обработки спутникового снимка Сан-Антонио. По часовой стрелке: входное изображение, полученный покров земли, настоящей покровы земли

Приложение Б. Примеры работы нейронной сети прогнозирования диапазона численности населения и последующего применения регрессионных моделей

(справочное)

Спутниковые снимки с наложенными тепловыми картами населения.

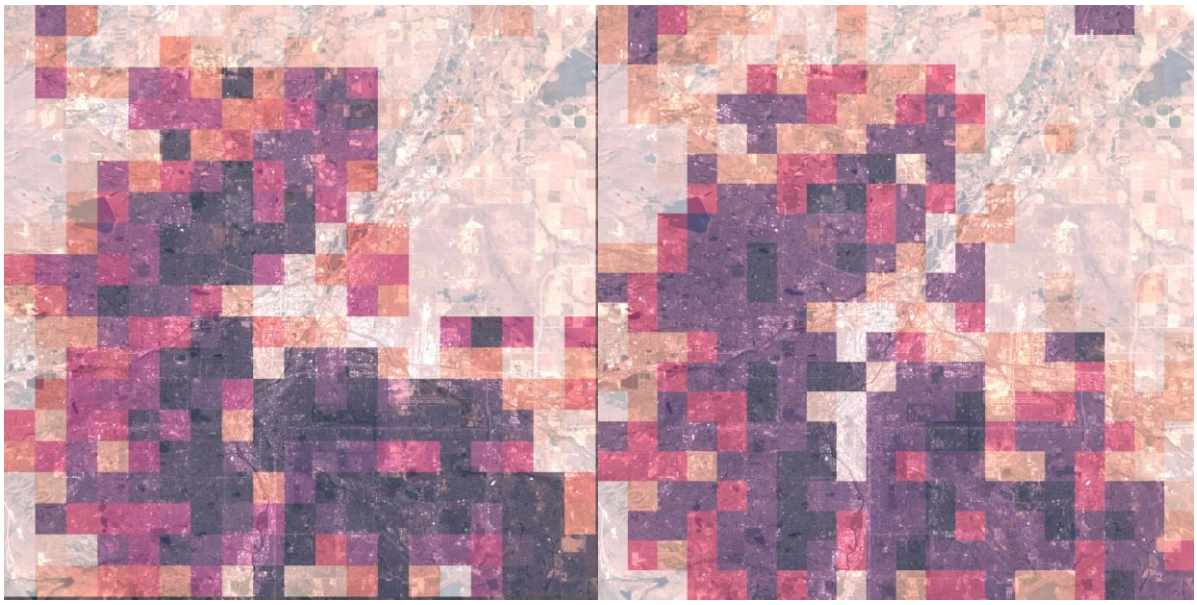

Рисунок Б.1. – Населения Денвера (слева) и полученное население (справа)

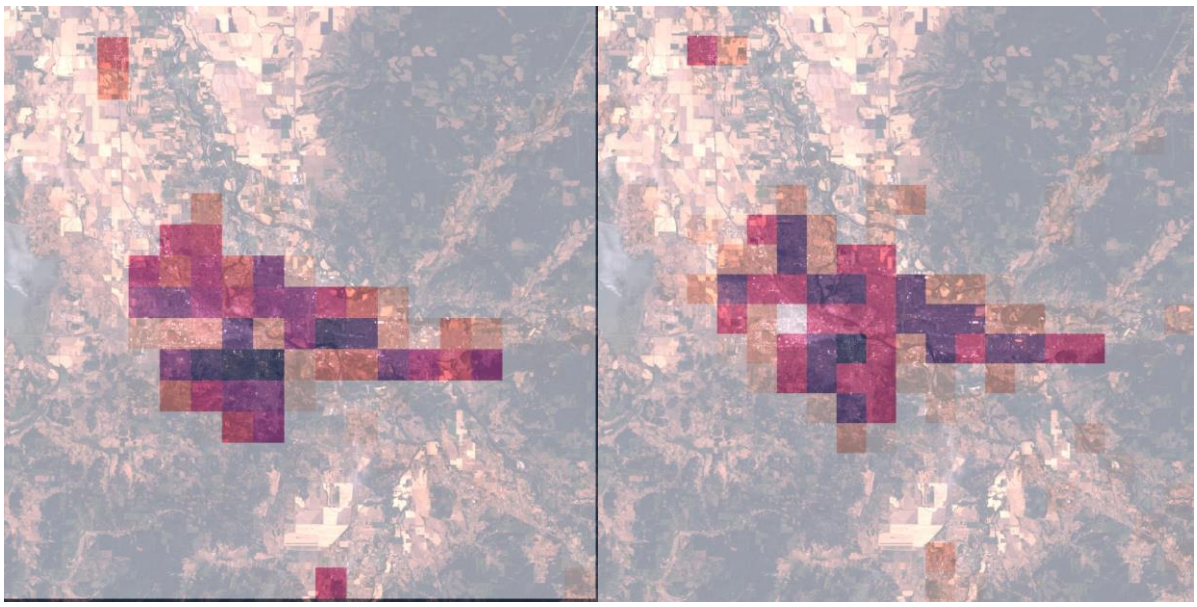

Рисунок Б.2. – Населения Юджина (слева) и полученное население (справа)

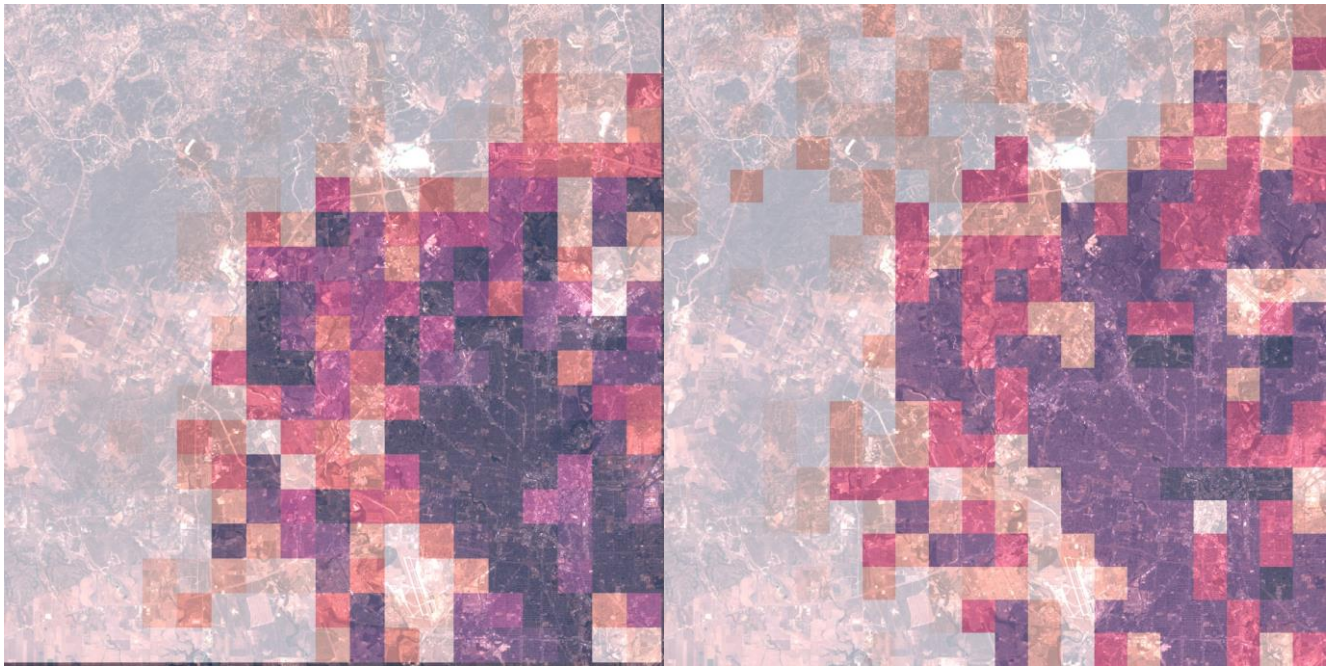

Рисунок Б.3. – Населения Сан-Антонио (слева) и полученное население (справа)

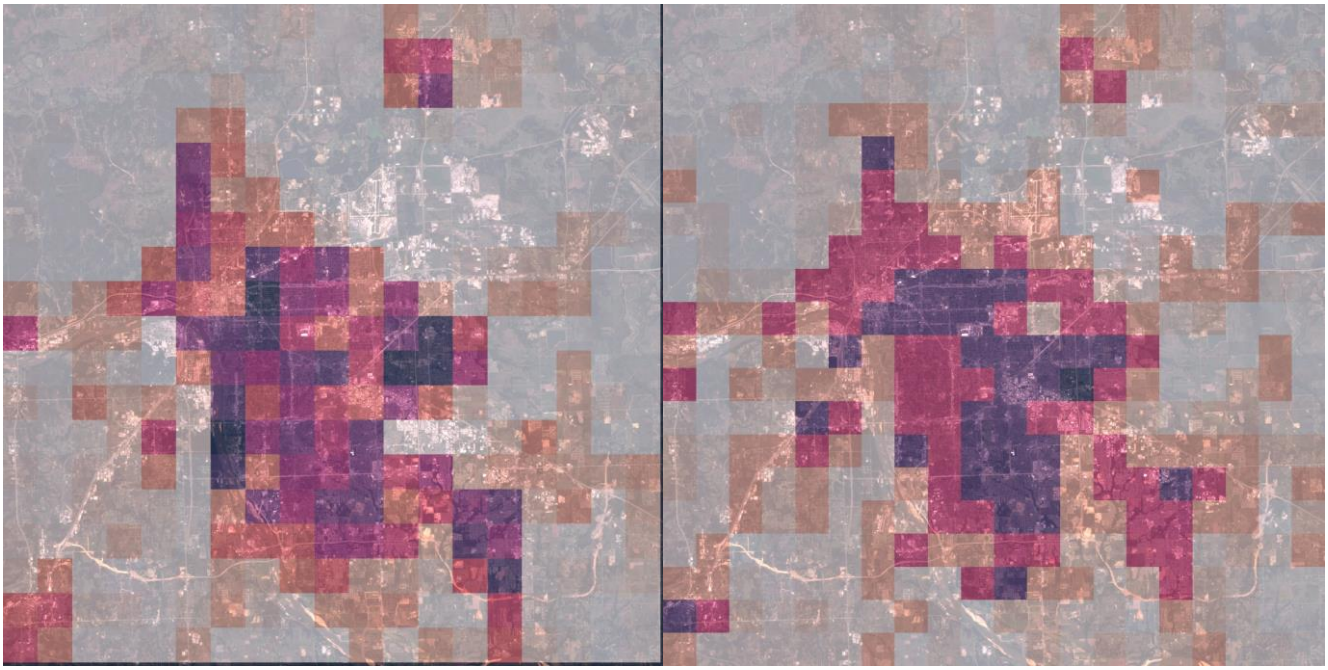

Рисунок Б.4. – Населения Талсы (слева) и полученное население (справа)

Приложение В. Программный код

(справочное)

Код проекта размещен в GitHub репозитории по URL: https://github.com/locsor/PopAprox.

Код, используемый для тренировки и тестирования модели нейронной сети U-Net++.

```
#Сверточный слой
def conv_layer(feat, filters, kernel_size, strides, training, reg = None, act_reg = None,
           name = None, trainable = True, padding = "same"):

   conv = tf.compat.v1.layers.Conv2D(filters=filters,
                       kernel_size=kernel_size,
                       strides=strides,
                       padding=padding,
                       name=name,
                       trainable=trainable,
                       kernel_regularizer=reg,
                       activity_regularizer=act_reg)(feat)
   act = tf.nn.leaky_relu(conv, alpha=0.1)
   bn = tf.compat.v1.layers.batch_normalization(act, center=True, scale=True,
                                                training=training, renorm=True)
   dropout = tf.compat.v1.layers.dropout(bn, rate = 0.5, training=training)
   return dropout

#Слой транспонированной свертки
def upsamp_layer(feat, filters, kernel_size, strides, training, reg = None,
                         act_reg = None, padding = "same"):
   transpose = tf.keras.layers.Conv2DTranspose(filters=filters,
                                kernel_size=kernel_size,
                                strides=strides,
                                padding=padding,
                                kernel_regularizer=reg,
                                activity_regularizer=act_reg)(feat)
   act = tf.nn.leaky_relu(transpose, alpha=0.1)
   bn = tf.compat.v1.layers.batch_normalization(act, center=True, scale=True,
                                                training=training, renorm=True)
   dropout = tf.compat.v1.layers.dropout(bn, rate = 0.5, training=training)
   return dropout

def pooling(feat):
   return tf.compat.v1.layers.MaxPooling2D(pool_size=[2, 2], strides=2)(feat)

def unet_pp2(features, labels, mode):
   training = False
```

```
features = tf.map_fn(lambda frame: tf.image.per_image_standardization(frame), features)
reg = None
act_reg = None
special_k = 5

conv000 = tf.compat.v1.layers.Conv2D(64,3,1,padding="same",name='block1_conv1',
                                     trainable = False, activation="relu")(features)
conv001 = tf.compat.v1.layers.Conv2D(64,3,1,padding="same",name='block1_conv2',
                                     trainable = False, activation="relu")(conv000)
pool1 = pooling(conv001)
conv100 = tf.compat.v1.layers.Conv2D(128,3,1,padding="same",name='block2_conv1',
                                     trainable = False, activation="relu")(pool1)
conv101 = tf.compat.v1.layers.Conv2D(128,3,1,padding="same",name='block2_conv2',
                                     trainable = False, activation="relu")(conv100)
pool2 = pooling(conv101)
conv200 = tf.compat.v1.layers.Conv2D(256,3,1,padding="same",name='block3_conv1',
                                     trainable = False, activation="relu")(pool2)
conv201 = tf.compat.v1.layers.Conv2D(256,3,1,padding="same",name='block3_conv2',
                                     trainable = False, activation="relu")(conv200)
conv202 = tf.compat.v1.layers.Conv2D(256,3,1,padding="same",name='block3_conv3',
                                     trainable = False, activation="relu")(conv201)
conv203 = tf.compat.v1.layers.Conv2D(256,3,1,padding="same",name='block3_conv4',
                                     trainable = False, activation="relu")(conv202)
pool3 = pooling(conv203)
conv300 = tf.compat.v1.layers.Conv2D(512,3,1,padding="same",name='block4_conv1',
                                     trainable = False, activation="relu")(pool3)
conv301 = tf.compat.v1.layers.Conv2D(512,3,1,padding="same",name='block4_conv2',
                                     trainable = False, activation="relu")(conv300)
conv302 = tf.compat.v1.layers.Conv2D(512,3,1,padding="same",name='block4_conv3',
                                     trainable = False, activation="relu")(conv301)
conv303 = tf.compat.v1.layers.Conv2D(512,3,1,padding="same",name='block4_conv4',
                                     trainable = False, activation="relu")(conv302)
pool4 = pooling(conv303)
conv400 = tf.compat.v1.layers.Conv2D(512,3,1,padding="same",name='block5_conv1',
                                     trainable = False, activation="relu")(pool4)
conv401 = tf.compat.v1.layers.Conv2D(512,3,1,padding="same",name='block5_conv2',
                                     trainable = False, activation="relu")(conv400)
conv402 = tf.compat.v1.layers.Conv2D(512,3,1,padding="same",name='block5_conv3',
                                     trainable = False, activation="relu")(conv401)
conv403 = tf.compat.v1.layers.Conv2D(512,3,1,padding="same",name='block5_conv4',
                                     trainable = False, activation="relu")(conv402)

upsamp01 = upsamp_layer(conv101, 64, 3, 2, training,reg,act_reg)
conv01 = conv_layer(tf.concat((upsamp01, conv001), axis = 3),64,special_k,1,training,reg,act_reg)
conv01 = conv_layer(conv01,64,3,1,training,reg,act_reg)
conv01 = conv_layer(conv01,64,3,1,training,reg,act_reg)

upsamp11 = upsamp_layer(conv203, 128, 3, 2, training,reg,act_reg)
conv11 = conv_layer(tf.concat((upsamp11, conv101), axis = 3),128,special_k,1,training,reg,act_reg)
conv11 = conv_layer(conv11,128,3,1,training,reg,act_reg)
conv11 = conv_layer(conv11,128,3,1,training,reg,act_reg)
```

```
upsamp02 = upsamp_layer(conv11, 64, 3, 2, training,reg,act_reg)
conv02 = conv_layer(tf.concat((upsamp02,conv001,conv01),axis = 3),64,special_k,1,
                                    training,reg,act_reg)
conv02 = conv_layer(conv02,64,3,1,training,reg,act_reg)
conv02 = conv_layer(conv02,64,3,1,training,reg,act_reg)

upsamp21 = upsamp_layer(conv303, 256, 3, 2, training,reg,act_reg)
conv21 = conv_layer(tf.concat((upsamp21, conv203), axis = 3),256,special_k,1,training,reg,act_reg)
conv21 = conv_layer(conv21,256,3,1,training,reg,act_reg)
conv21 = conv_layer(conv21,256,3,1,training,reg,act_reg)

upsamp12 = upsamp_layer(conv21, 128, 3, 2, training,reg,act_reg)
conv12 = conv_layer(tf.concat((upsamp12, conv101, conv11), axis = 3),128,special_k,1,
                                    training,reg,act_reg)
conv12 = conv_layer(conv12,128,3,1,training,reg,act_reg)
conv12 = conv_layer(conv12,128,3,1,training,reg,act_reg)

upsamp03 = upsamp_layer(conv12, 64, 3, 2, training,reg,act_reg)
conv03 = conv_layer(tf.concat((upsamp03, conv001, conv01, conv02), axis = 3),64,special_k,1,
                                    training,reg,act_reg)
conv03 = conv_layer(conv03,64,3,1,training,reg,act_reg)
conv03 = conv_layer(conv03,64,3,1,training,reg,act_reg)

upsamp31 = upsamp_layer(conv403, 512, 3, 2, training,reg,act_reg)
conv31 = conv_layer(tf.concat((upsamp31, conv303), axis = 3),512,special_k,1,training,reg,act_reg)
conv31 = conv_layer(conv31,512,3,1,training,reg,act_reg)
conv31 = conv_layer(conv31,512,3,1,training,reg,act_reg)

upsamp22 = upsamp_layer(conv31, 256, 3, 2, training,reg,act_reg)
conv22 = conv_layer(tf.concat((upsamp22, conv203, conv21), axis = 3),256,special_k,1,
                                    training,reg,act_reg)
conv22 = conv_layer(conv22,256,3,1,training,reg,act_reg)
conv22 = conv_layer(conv22,256,3,1,training,reg,act_reg)

upsamp13 = upsamp_layer(conv22, 128, 3, 2, training,reg,act_reg)
conv13 = conv_layer(tf.concat((upsamp13, conv101, conv11, conv12), axis = 3),128,special_k,1,
                                    training,reg,act_reg)
conv13 = conv_layer(conv13,128,3,1,training,reg,act_reg)
conv13 = conv_layer(conv13,128,3,1,training,reg,act_reg)

upsamp04 = upsamp_layer(conv13, 64, 3, 2, training,reg,act_reg)
conv04 = conv_layer(tf.concat((upsamp04, conv001, conv01, conv02, conv03), axis = 3),64,
                                    special_k,1,training,reg,act_reg)
conv04 = conv_layer(conv04,64,3,1,training,reg,act_reg)
conv04 = conv_layer(conv04,64,3,1,training,reg,act_reg)

decoder1 = tf.compat.v1.layers.Conv2D(filters=6,
                    kernel_size=1,
                    strides=1,
                    padding="same",
                    activation="sigmoid")(conv01)
```

```
    decoder2 = tf.compat.v1.layers.Conv2D(filters=6,
                        kernel_size=1,
                        strides=1,
                        padding="same",
                        activation="sigmoid")(conv02)

    decoder3 = tf.compat.v1.layers.Conv2D(filters=6,
                        kernel_size=1,
                        strides=1,
                        padding="same",
                        activation="sigmoid")(conv03)

    decoder4 = tf.compat.v1.layers.Conv2D(filters=6,
                        kernel_size=1,
                        strides=1,
                        padding="same",
                        activation="sigmoid")(conv04)

    votes = tf.stack([decoder1, decoder2, decoder3, decoder4], -1)
    probs = tf.reduce_mean(votes, -1)
    classes = tf.argmax(input=probs, axis=-1)

    predictions = {
      "classes": classes,
      "probabilities": votes,
    }

    if mode == tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)
```

Код, используемый для тренировки и тестирования нейронной сети прогнозирования диапазона численности населения.

```
def conv_layer(feat, filters, kernel_size, strides, training, reg = None, act_reg = None, name = None,
trainable = True, padding = "same", relu = True):
    conv = tf.compat.v1.layers.Conv2D(filters=filters,
                        kernel_size=kernel_size,
                        strides=strides,
                        padding=padding,
                        name=name,
                        trainable=trainable,
                        kernel_regularizer=reg,
                        activity_regularizer=act_reg)(feat)

    bn = tf.compat.v1.layers.batch_normalization(conv, center=True, scale=True, training=training)
```

```
    if relu:
        return tf.nn.leaky_relu(bn, alpha=0.1)
    else:
        return bn


def pooling(feat):
    return tf.compat.v1.layers.MaxPooling2D(pool_size=3, strides=2)(feat)

def resnet_block(feat, filters, kernel_size, strides, training, reg, relu = True, trainable = True):
    conv = tf.compat.v1.layers.Conv2D(filters,kernel_size,strides,padding="same",
                                                    trainable = trainable, kernel_regularizer=reg)(feat)
    bn = tf.compat.v1.layers.batch_normalization(conv, training=training, trainable = trainable)

    if relu:
        return tf.nn.leaky_relu(bn, alpha=0.1)
    else:
        return bn

def block(feat, filters, training, reg=None, stride = 1):

    conv0 = resnet_block(feat, filters, 1, stride, training, reg)
    conv1 = resnet_block(conv0, filters, 3, 1, training, reg)

    if feat.shape[-1] == filters*4:
        conv2 = resnet_block(conv1, filters*4, 1, 1, training, reg, relu = False) + feat
    else:
        conv2 = resnet_block(conv1, filters*4, 1, 1, training, reg, relu = False) + \
                resnet_block(feat, filters*4, 1, stride, training, reg, relu = False)
    return tf.nn.leaky_relu(conv2, alpha=0.1)

def sub_model(features, training, l2_reg = None, ):
    conv0 = resnet_block(features, 64, 7, 1, training, l2_reg)
    pool0 = pooling(conv0)

    block0 = block(pool0, 64, training, l2_reg, stride = 2)
    block1 = block(block0, 64, training, l2_reg)
    block2 = block(block1, 64, training, l2_reg)

    block3 = block(block2, 128, training, l2_reg, stride = 2)
    block4 = block(block3, 128, training, l2_reg)
    block5 = block(block4, 128, training, l2_reg)
    block6 = block(block5, 128, training, l2_reg)
    block_special = block(block6, 256, training, l2_reg, stride = 2)

    for i in range(22):
        block_special = block(block_special, 256, training, l2_reg)

    block13 = block(block_special, 512, training, l2_reg, stride = 2)
    block14 = block(block13, 512, training, l2_reg)
    block15 = block(block14, 512, training, l2_reg)
```

```
    return block15


def resnet(features, labels, mode):
    training = False
    if mode == tf.estimator.ModeKeys.TRAIN:
        training = True

    feature1 = features[..., :3]
    feature2 = tf.expand_dims(features[..., 3], -1)
    features1 = tf.map_fn(lambda frame: tf.image.per_image_standardization(frame), feature1)
    features2 = tf.map_fn(lambda frame: tf.image.per_image_standardization(frame), feature2)

    l2_reg = None

    out1 = sub_model(features1, training, l2_reg)
    out2 = sub_model(features2, training, l2_reg)
    avg1 = tf.keras.layers.GlobalAveragePooling2D()(out1)
    avg2 = tf.keras.layers.GlobalAveragePooling2D()(out2)
    merge = tf.concat((avg1, avg2), axis = -1)

    out = tf.compat.v1.layers.dense(merge, 2048)
    out_bn = tf.compat.v1.layers.batch_normalization(out, center=True, scale=True, training=training)
    out_relu = tf.nn.relu(out_bn)
    logits = tf.keras.layers.Dense(11, activation=None)(out_relu)

    predictions = {
      "classes": tf.argmax(input=tf.nn.softmax(logits), axis=1),
      "probabilities": tf.nn.softmax(logits),
      "feats": out_relu
    }

    return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)
```