

КАЗАХСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ
имени АЛЪ-ФАРАБИ

РЕКОМЕНДАЦИИ
ПО КЛАССИФИКАЦИИ
ЗЕМЛЕПОЛЬЗОВАНИЯ И
ЗЕМЕЛЬНОГО ПОКРОВА
(LAND USE AND LAND COVER)

Алматы
«ESKoPrint»
2026

УДК 332.33

ББК 65.32-5

Р 36

*Рекомендована к изданию Ученым советом
КазНУ им. аль-Фараби (протокол № 11 от 24.04.2026 г.)*

Рецензенты:

Дюсембеков З.Д., доктор сельскохозяйственных наук, профессор кафедры земельного кадастра, НАО «Казахский агротехнический исследовательский университет им. С. Сейфуллина», академик НАН РК.

Бексентова Р.Т., доктор географических наук, профессор кафедры картографии и геоинформатики, НАО «Казахский национальный университет им аль-Фараби».

Авторы:

*О.Ә. Әліпбеки, Д.К. Рахимов, П.П. Гроссул,
Ч.А. Аліпбекова, Д. Кенжебек, Қ. Аугамбаев, М.М. Алиев*

Р 36 Рекомендации по классификации землепользования и земельного покрова / коллектив авторов. – Алматы: ESKoPrint, 2026. – 196 с.

ISBN 978-601-04-7644-8

Казахстан после реализации первого этапа Национальной инфраструктуры пространственных данных (НИПД) вошел в элиту стран в области системного развития цифровизации. Однако, мировой динамизм, требует постоянного совершенствования НИПД, включая её отраслевых компонентов, одним из которых является Аграрная ИПД Республики Казахстан (АгроИПД РК). Как правило земли агропромышленного комплекса (АПК) и сельских территории (СТ) часто меняют свои функции и характеристики под влиянием антропогенных и природных факторов. Поэтому систематическая классификация земель по разным классам, с целью знания тренда развития землепользования в АПК и СТ является первоочередной необходимостью для оценки их устойчивого развития. Настоящие рекомендации направлены на выполнение классификации землепользования и земельного покрова (Land use and Land Cover - LULC) на основе пространственных данных с использованием геоинформационных технологий, методов дистанционного зондирования Земли, машинного и глубокого обучения (искусственный интеллект). Они охватывают этапы проектирования методики классификации, подготовки и отбора исходных спутниковых данных, применения алгоритмов анализа и моделирования, оценки точности результатов, а также подготовка необходимых скриптов в полном объеме. В результате специалисты в области сельского хозяйства, управления земельными, лесными, водными ресурсами, экологии и других смежных областей смогут интегрировать данные о LULC в свои рабочие процессы для повышения эффективности мониторинга, контроля, планирования и устойчивого развития АПК и СТ.

Рекомендации разработаны по результатам исследований в рамках научно-технической программы «Формирование аграрной инфраструктуры пространственных данных Республики Казахстан по технологиям и принципам 2.0», ИРН: BR 22886730 по линии программно-целевого финансирования Министерства сельского хозяйства Республики Казахстан.

УДК 332.33

ББК 65.32-5

ISBN 978-601-04-7644-8

© Коллектив авторов, 2026

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

АгроИПД	Аграрная инфраструктуры пространственных данных
АПК	Агропромышленный комплекс
БПЛА	Беспилотный летательный аппарат
ГИС	Геоинформационные системы
ДЗЗ	Дистанционное зондирование Земли
ПВД	Пространственно-временные данные
ПД	Пространственные данные
СТ	Сельские территории
AI	Artificial Intelligent
CART	Classification and Regression Trees
CNN	Convolutional Neural Network
EVI	Enhanced Vegetation Index
GAN	Generative Adversarial Nnetworks
GEE	Google Earth Engine
GMM	Gaussian Mixture Models
LDA	Linear Discriminant Analysis
LULC	Land Use and Land Cover
MLC	Maximum Likelihood Classification
MLP	Multilayer Perception
NDVI	Normalized Difference Vegetation Index
QGIS	Quantum Geoinformation System
RF	Random Forest
ROI	Region of interest
SCP	Semi-Automatic Classification Plugin
SVM	Support Vector Machine
UI	User Interface
WGS 1984	World Geodetic System 1984

НОРМАТИВНЫЕ ДОКУМЕНТЫ

Закон Республики Казахстан «О геодезии, картографии и пространственных данных», от 21 декабря 2022 года № 166-VII ЗРК.

Об утверждении Концепции развития агропромышленного комплекса Республики Казахстан на 2021–2030 годы. Постановление Правительства Республики Казахстан от 30 декабря 2021 года № 960.

Об утверждении Концепции цифровой трансформации, развития отрасли информационно-коммуникационных технологий и кибербезопасности на 2023–2029 годы. Постановление Правительства Республики Казахстан от 28 марта 2023 года № 269.

Земельный кодекс Республики Казахстан. 20 июня 2003 года № 442-I ЗРК,

Лесной кодекс Республики Казахстан, от 8 июля 2003 года № 477 ЗРК.

Водный кодекс Республики Казахстан от 9 июля 2003 года № 481 ЗРК, основополагающий документ.

Экологический кодекс Республики Казахстан от 2 января 2021 года № 400-VI ЗРК.

Закон Республики Казахстан «О космической деятельности», с изменениями и дополнениями по состоянию на 11.04.2019 г. ЗРК.

Закон Республики Казахстан от 24 ноября 2015 года № 418-V ЗРК. «Об информатизации»

Национальный план развития Республики Казахстана до 2029 года. Астана, 2024 год. Указ Президента Республики Казахстан от 30 июля 2024 года № 611.

О науке и технологической политике. Закон Республики Казахстан от 1 июля 2024 года № 103-VIII ЗРК.

Об искусственном интеллекте. Закон Республики Казахстан от 17 ноября 2025 года № 230-VIII ЗРК.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	7
Глава 1. ПАРАМЕТРИЧЕСКИЕ МОДЕЛИ КЛАССИФИКАЦИИ	10
1.1 Метод максимального правдоподобия (Maximum Likelihood Classification - MLC)	11
1.1.1 Инструкция по классификации методом MLC	13
1.2 Метод минимального расстояния (Minimum Distance - MD)	18
1.2.1 Инструкция по классификации методом MD	21
1.3 Спектральный угловой картограф (Spectral Angle Mapper - SAM)	25
1.3.1 Инструкция по классификации методом SAM	26
Глава 2. КЛАССИФИКАЦИЯ С ИСПОЛЬЗОВАНИЕМ МАШИННОГО ОБУЧЕНИЯ (Machine Learning - ML)	31
2.1 Характеристика алгоритмов, основанных на Machine learning	31
2.2 Инструкция по классификации LULC моделями Machine learning	33
2.2.1 Классификация LULC на платформе Google Earth Engine	34
2.2.2 Коды классификаторов машинного обучения	41
2.3 Ограничения алгоритмов машинного обучения	56
Глава 3. КЛАССИФИКАЦИЯ С ИСПОЛЬЗОВАНИЕМ ГЛУБОКОГО ОБУЧЕНИЯ (Deep Learning - DL)	58
3.1 Сверточные нейронные сети (Convolutional Neural Network - CNN)	59
3.3.1 Инструкция по классификации LULC с использованием CNN	60
3.2 Преимущества, вызовы и ограничения, перспективные направления глубокого обучения	171

Глава 4. СРАВНЕНИЕ МОДЕЛЕЙ ДЛЯ КЛАССИФИКАЦИИ LULC	173
ЗАКЛЮЧЕНИЕ	185
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ	189
ГЛОССАРИЙ.....	192

ВВЕДЕНИЕ

Классификация землепользования и земельного покрова (Land use and Land cover -LULC) представляет собой важный инструмент для понимания пространственного распределения природных и антропогенных объектов, их динамики и влияния на окружающую среду. Эта дисциплина лежит в основе многих областей науки и практики, включая сельское хозяйство, управление земельными, водными и лесными ресурсами, градостроительство, природоохранную деятельность, прогнозирование изменений и оценку воздействия человека на природные экосистемы. Опираясь на методы геоинформатики и дистанционного зондирования, классификация LULC позволяет создать детальные пространственно-временные цифровые карты конкретных территорий, которые необходимы для стратегического планирования и принятия управленческих решений.

Исторически классификация LULC основывалась на визуальном анализе аэрофотоснимков и полевых обследований, что требовало значительных временных и трудовых затрат. Однако с развитием технологий дистанционного зондирования Земли (ДЗЗ) и геоинформационных систем (ГИС) этот процесс претерпел революционные изменения. Современные спутниковые системы предоставляют изображения с высоким пространственным, спектральным и временным разрешением, что позволяет исследователям более точно различать объекты на поверхности Земли. В сочетании с передовыми методами обработки данных, такими как машинное обучение и глубокое обучение (нейронные сети - искусственный интеллект), это открывает новые возможности для автоматизированной классификации LULC с высокой точностью и скоростью.

Современные подходы к классификации LULC включают широкий спектр методов, от традиционных статистических моделей до передовых алгоритмов машинного и глубокого обучения. Параметрические модели, такие как метод максимального правдоподобия, основаны на строгих статистических допущениях и требуют предварительного знания распределения данных. Непараметрические модели, включая деревья решений, случайные леса и поддерживающие векторные машины, обеспечивают

большую гибкость, адаптируясь к сложным структурам данных. В последние годы большее внимание привлекают методы глубокого обучения, такие как сверточные нейронные сети (Convolutional Neural Network - CNN), которые демонстрируют выдающиеся результаты в обработке изображений и автоматической классификации LULC.

Развитие технологий в области классификации LULC продолжается быстрыми темпами. Внедрение облачных вычислений и интеграция данных из различных источников, включая спутниковые снимки, аэрофотосъемку, данные беспилотных летательных аппаратов (БПЛА) и полевые измерения, позволяют получать все более точные и детализированные карты LULC. Искусственный интеллект и автоматизация анализа данных сокращают временные затраты и повышают воспроизводимость результатов, что делает процесс классификации более доступным и эффективным.

Классификация LULC является неотъемлемым элементом современного управления сельским хозяйством, земельными, водными, лесными и другими природными ресурсами с целью их устойчивого развития. Ее применение затрагивает широкий круг задач – от глобального мониторинга изменения климата до локального землепользования в отдельных регионах. Современные технологии обеспечивают новые возможности для исследования и анализа, позволяя получать точные и надежные данные о состоянии окружающей среды. В условиях нарастающих экономических и экологических проблем, а также роста населения планеты разработка и совершенствование методов классификации LULC становится одной из приоритетных задач научного сообщества и практиков в области сельского хозяйства, управления земельными, водными, лесными и другими природными ресурсами.

Главной целью классификации LULC является систематическое и однозначное описание физического покрытия земной поверхности (Земельный покров - Land Cover) с одновременным выявлением ее функционального, хозяйственного или социально-экономического назначения (Землепользование - Land Use) для решения конкретных прикладных или научных задач, позволяющий видеть не только статичную картинку, но и векторы изменений для управления земельными, лесными, водными и другими ресурсами и прогнозирования. Применительно к созданию аграрной

инфраструктуры пространственных данных (АгроИПД) версии 2.0, главная цель классификации LULC смещается от простого описания территории к созданию семантически согласованной, многоуровневой и "живой" модели, которая непосредственно питает цифровые двойники агроэкосистем и системы поддержки принятия решений.

Задачи классификации LULC для АгроИПД 2.0 охватывают полный цикл: от интеграции мультиспектральных данных и применения алгоритмов машинного и глубокого обучения до обеспечения высокой точности ($F1\text{-score}$ для аграрных классов > 0.75) и мониторинга динамики с прогнозированием сценариев развития. Ключевое отличие от традиционных подходов — ориентация на аграрно-специфичные классы и интеграция с социально-экономическими и экологическими и моделями для поддержки принятия управленческих решений.

Целью данной рекомендации является довести читателя до непосредственного применения отдельных алгоритмов классификации LULC для решения собственных проблем, которые возникают в ходе использования земельных, лесных, водных и других ресурсов на конкретной территории.

Задачи нашего труда направлены на практическое использование алгоритмов трех моделей классификации LULC: параметрических, методов машинного и глубокого обучения.

При необходимости, заинтересованный читатель может получить все скрипты в Word формате, написав письмо на следующий E-mail: Alipbeki.Onggarbek@kaznu.kz

Глава 1. ПАРАМЕТРИЧЕСКИЕ МОДЕЛИ КЛАССИФИКАЦИИ

Параметрические модели классификации представляют собой один из фундаментальных подходов в статистическом анализе, основанный на предположении о распределении данных. Они характеризуются фиксированным числом параметров, которые определяют форму распределения классов в пространстве признаков. В отличие от непараметрических методов, параметрические модели требуют априорных предположений о структуре данных и часто основываются на статистических законах. Параметрические алгоритмы основаны на статистических моделях и предположениях о распределении данных, что делает их особенно привлекательными для многих исследователей и практиков. Они обладают рядом преимуществ:

- простота - алгоритмы относительно просты в реализации и понимании;

- интерпретируемость - результаты легко интерпретировать, что важно для принятия решений;

- эффективность - при правильных предположениях о распределении данных, эти методы могут быть очень эффективными;

- фиксированное число параметров – количество параметров не зависит от размера обучающей выборки, что упрощает обучение модели;

- предположение о распределении данных – чаще всего используется нормальное (гауссово) распределение признаков внутри каждого класса;

- высокая вычислительная эффективность – за счет ограниченного числа параметров такие модели обучаются быстрее, чем сложные алгоритмы, например, глубокие нейронные сети;

- линейная и квадратичная разделимость – большинство параметрических моделей строят линейные или квадратичные границы между классами.

Наиболее распространенными параметрическими алгоритмами являются: метод максимального правдоподобия, метод минимального расстояния, расстояние Махаланобиса, спектральный угловой картограф, спектральный корреляционный картограф, параллельный метод.

1.1. Метод максимального правдоподобия (Maximum Likelihood Classification - MLC)

MLC – один из наиболее распространенных параметрических алгоритмов классификации LULC. Он широко применяется в классификации данных ДЗЗ благодаря своей математической строгости и высокой точности. Однако его эффективность напрямую зависит от качества предварительной оценки статистических параметров классов, что требует тщательной калибровки обучающих данных. В частности, метод предполагает, что все классы следуют многомерному нормальному распределению, что не всегда соответствует реальным данным. В случае сложных, неоднородных или сильно перекрывающихся классов, точность MLC может снижаться, поскольку реальные распределения могут отклоняться от нормальных.

Еще одной важной особенностью MLC является учет ковариационной матрицы, которая позволяет модели различать классы даже при частичном совпадении значений признаков. Чем больше различие между ковариационными матрицами классов, тем выше вероятность корректной классификации. Однако если ковариационные матрицы классов схожи или если данные содержат значительный уровень шума, MLC может демонстрировать переклассификацию пикселей и снижение точности. Кроме того, метод требует большого количества размеченных данных для корректной оценки параметров распределения, что может усложнять его применение в случаях ограниченного объема обучающих выборок.

Этот метод основан на вычислении вероятности принадлежности пикселя к определенному классу с использованием статистических параметров распределения значений яркости (рисунок 1.1).

График на рисунке 1 иллюстрирует функцию правдоподобия $L(\theta|data)$ для оценки параметра θ нормального распределения методом максимального правдоподобия. Ось абсцисс представляет возможные значения параметра θ . Ось ординат показывает значения функции правдоподобия $L(\theta|data)$. Красная пунктирная линия обозначает истинное значение параметра θ_{true} . Зеленая пунктирная линия указывает на оценку максимального правдоподобия $MLE\theta$.

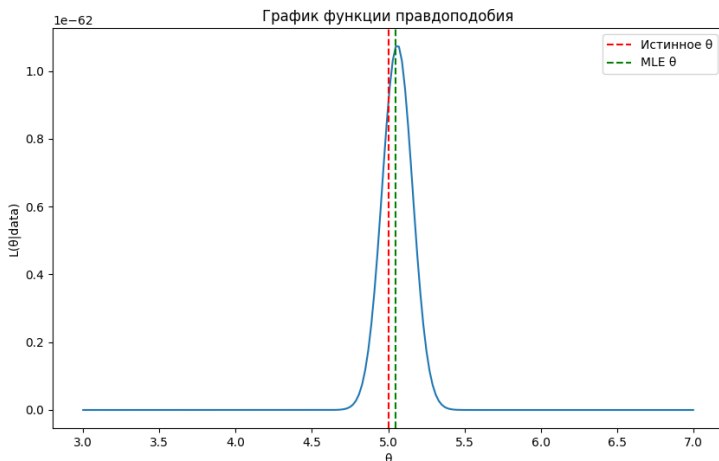


Рисунок 1.1 – График функции максимального правдоподобия

Пик функции правдоподобия соответствует значению $MLE\theta$, которое максимизирует вероятность наблюдения данных. Близость $MLE\theta$ к θ_{true} демонстрирует эффективность метода максимального правдоподобия в оценке параметров распределения.

Алгоритм MLC предполагает, что статистические характеристики классов подчиняются многомерному нормальному распределению. Для каждого класса вычисляется вектор средних значений и ковариационная матрица. Затем для каждого пикселя рассчитывается вероятность его принадлежности к каждому классу, и пиксель относится к классу с максимальной вероятностью. Математически это можно выразить следующим образом:

$$g_i(x) = \ln p(w_i) - \frac{1}{2} \ln |\Sigma_i| - \frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) \quad (1)$$

где $g_i(x)$ – дискриминантная функция для класса i , $p(w_i)$ – априорная вероятность класса i , Σ_i – ковариационная матрица класса i , μ_i – вектор средних значений класса i , x – вектор признаков классифицируемого пикселя.

MLC остается одним из основных параметрических алгоритмов, широко применяемых в классификации LULC. В сочетании с предварительной обработкой данных, уменьшением размерности признакового пространства и улучшением алгоритмов выбора

обучающих выборок можно значительно повысить точность MLC и адаптировать его к сложным сценариям классификации. В современных исследованиях MLC часто комбинируют с непараметрическими методами (например, деревьями решений и методами глубокого обучения) для улучшения результатов классификации и компенсации его ограничений.

1.1.1 Инструкция по классификации методом максимального правдоподобия

Требование для проведения классификации:

- Установленное ПО ArcGIS/ArcMap (версия 10.3 и выше).
- Загруженные спутниковые снимки (Landsat, Sentinel и т.д.).
- Наличие обучающих полигонов (Training Samples) для

каждого класса LULC.

Шаг - 1. Подготовка данных

Импорт снимка

- Откройте ArcMap.
- Вставьте спутниковый растр в рабочую область (File → Add Data → Add Raster Data).
- Убедитесь, что изображение корректно отображается (рисунок 1.2).

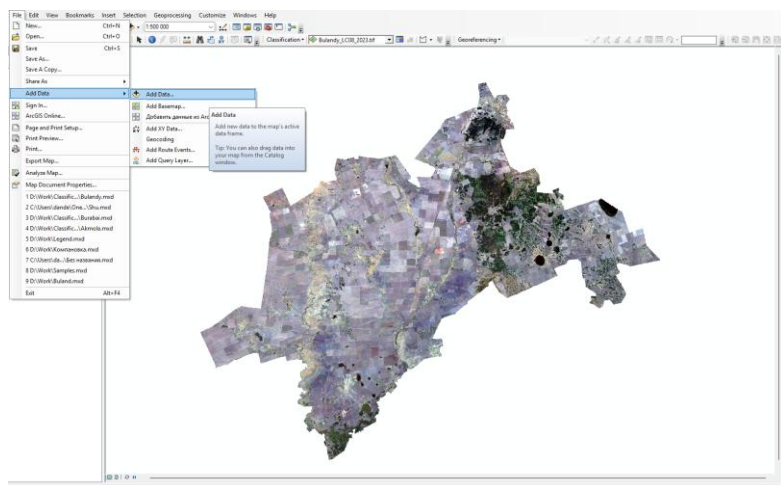


Рисунок 1.2 – Путь для интеграции спутникового растра в рабочую область

Создание фрейма данных:

- Перейдите в Data Frame Properties → Coordinate System.
- Установите правильную проекцию, например WGS 1984 UTM Zone соответствующего региона (рисунок 1.3).

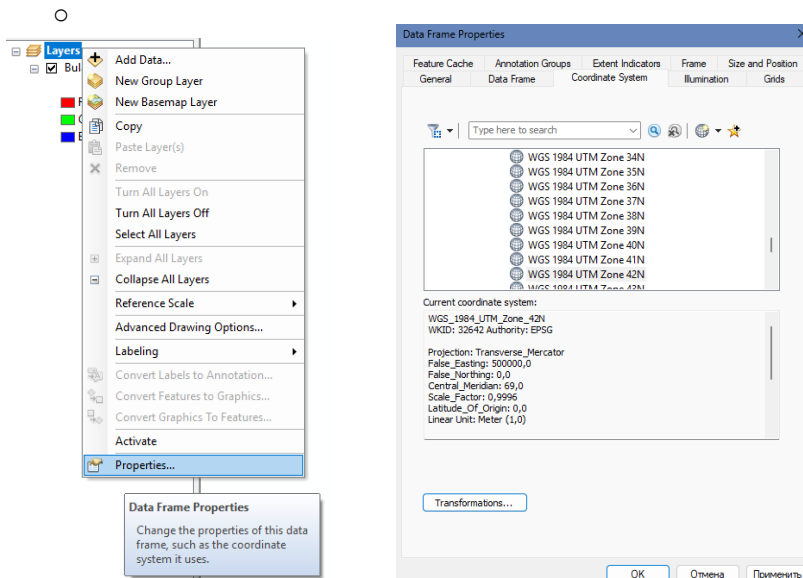


Рисунок 1.3 – Путь для установки координатной сети

Шаг - 2. Создание обучающих образцов (Training Samples)

Откройте Image Classification Toolbar:

- Включите: Customize → Toolbars → Image Classification.
- Добавьте новый слой классов (Training Sample Manager) (рисунок 1.4).

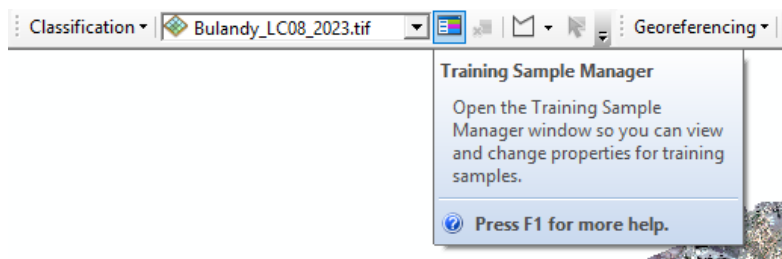


Рисунок 1.4 – Создание слоя классов для классификации

- Используя инструмент Draw Polygon, вручную обведите участки каждого класса (лес, вода, застройка и т.д.) (рисунок 1.5).

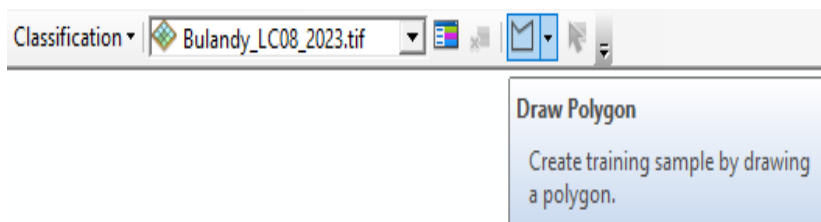


Рисунок 1.5 – Рисование полигонов для слоя классов

- Назовите классы (Cropland, Water, Urban и т.д.) (рисунок 1.6).

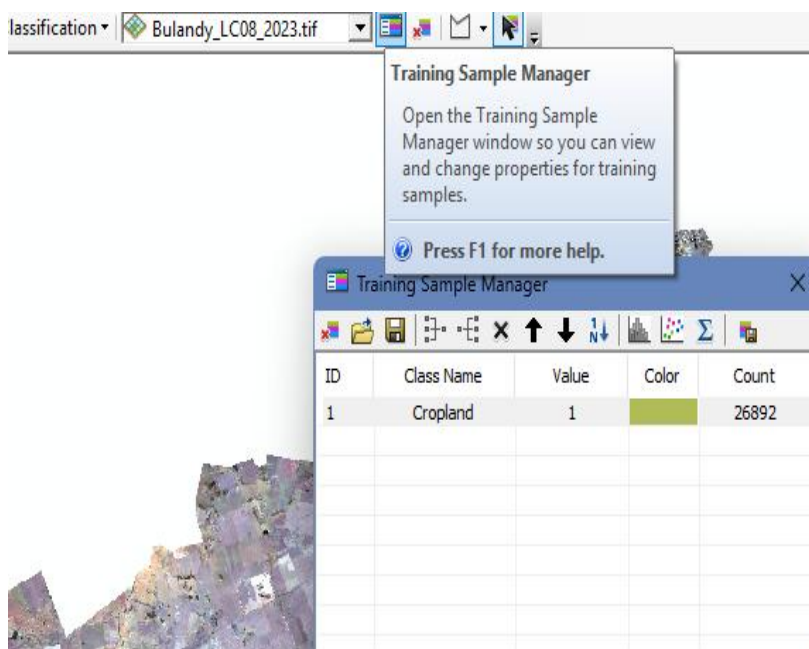


Рисунок 1.6 – Конфигурация полученного слоя класса

○ Сохраните сигнатурный файл: Training Sample Manager
→ Export Signature File.

Шаг – 3. Классификация методом Maximum Likelihood

• Перейдите в ArcToolbox → Spatial Analyst Tools → Multivariate → Maximum Likelihood Classification (рисунок 1.7).

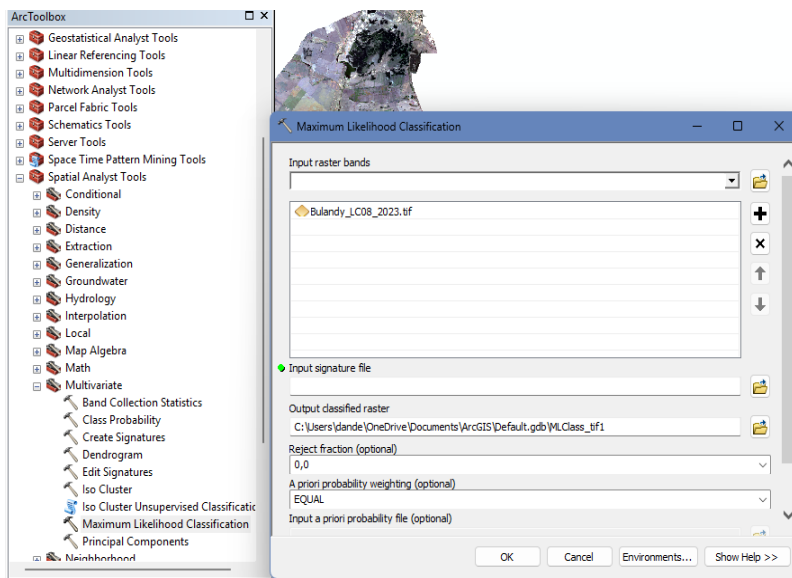


Рисунок 1.7 - Путь до классификатора

• В открывшемся окне укажите:

- Input raster bands: ваш многоканальный снимок (например, RGB или все каналы Landsat).
- Input signature file: ранее сохранённый .gsg файл.
- Output classified raster: имя и путь к результирующему классифицированному изображению.

• Нажмите ОК.

Шаг - 4. Визуализация результатов

В ArcMap добавится классифицированный растр. Задайте цветовую схему (Symbology) → Unique Values, выбрав нужные цвета для каждого класса (рисунок 1.8).

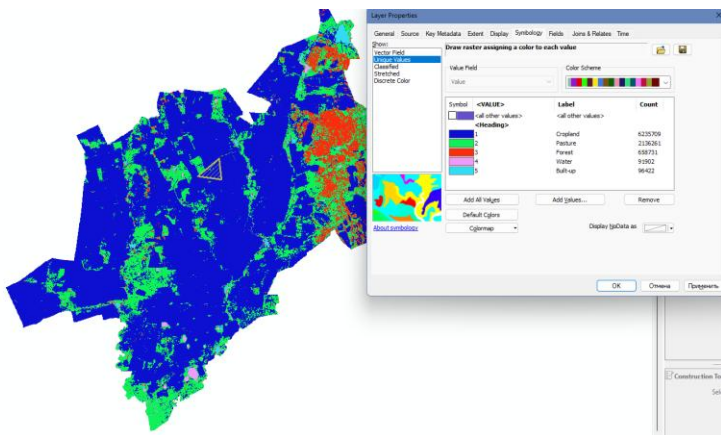


Рисунок 1.8 - Изменение цветовой схемы в свойстве раstra

Шаг - 5. Оценка точности классификации

Сбор тестовых данных (ground truth) – полигональные/точечные данные из GPS или визуального распознавания.

Перейдите в ArcToolbox → Spatial Analyst Tools → Multivariate → Confusion Matrix (рисунок 1.9).

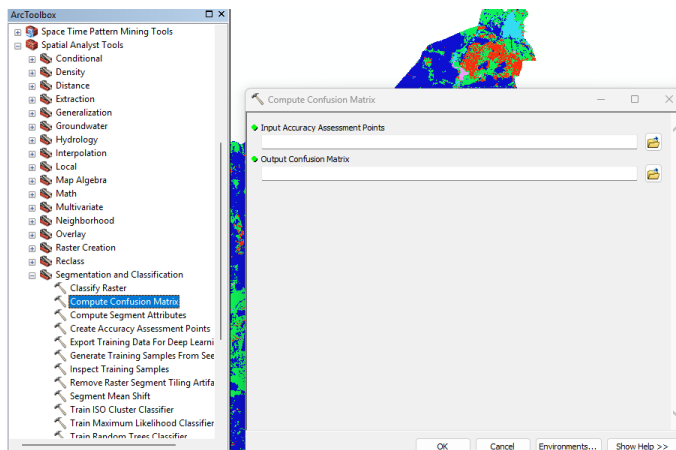


Рисунок 1.9 - Путь до инструмента (Compute Confusion Matrix) для получения матрицы ошибок

Введите:

- Input raster: классифицированный снимок.
- Ground truth layer: слой верификации.
- Укажите поле классов.

Получите матрицу ошибок, Overall Accuracy, User's Accuracy, Producer's Accuracy, и Kappa Index.

Результат классификации (рисунок 1.10):

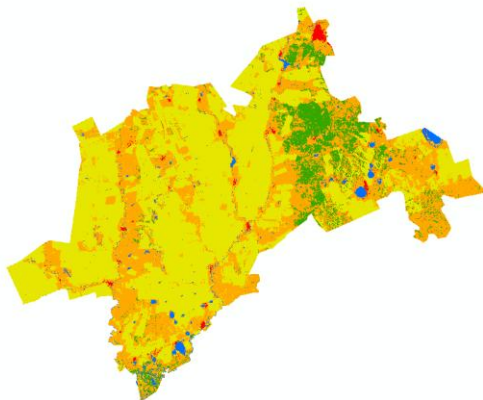


Рисунок 1.10 - Классифицированное изображение Буландинского района методом Maximum Likelihood

1.2 Метод минимального расстояния (Minimum Distance - MD)

Minimum Distance – простой и быстрый алгоритм классификации LULC. Он основан на вычислении евклидова расстояния между вектором признаков пикселя и центроидами классов. Для каждого класса вычисляется центрoид (среднее значение признаков). Затем для каждого пикселя рассчитывается евклидово расстояние до центрoида каждого класса, и пиксель относится к классу с минимальным расстоянием. На рисунке 1.11 изображена классификация данных методом минимальных расстояний до средних значений.

График включает следующие элементы:

1) Точки данных: Точки данных распределены по шести классам:

- водные объекты (голубой)
- населенные пункты (красный)
- лес (зеленый)
- пахотные земли (желтый)
- почвы (бордовый)

2) Центроиды классов: Центроиды каждого класса обозначены синими квадратами с черной окантовкой.

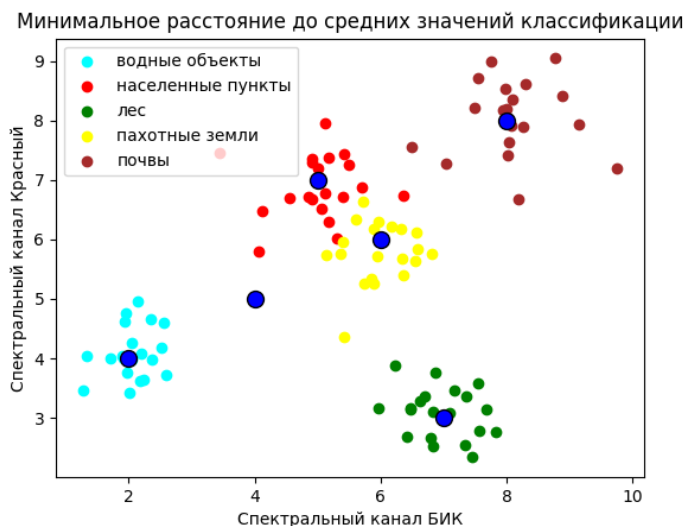


Рисунок 1.11 – Минимальное расстояние до средних значений классификации

3) Минимальные расстояния: Желтые пунктирные линии показывают минимальные расстояния от каждой точки данных до ближайшего центроида.

4) Оси координат: Ось X соответствует спектральному каналу БИК, а ось Y – спектральному каналу «Красный».

График иллюстрирует процесс классификации, при котором каждая точка данных относится к классу, чье среднее значение (центроид) находится на минимальном расстоянии от этой точки. Математически это можно выразить следующим образом:

$$D = \sqrt{\sum_{i=1}^n (x_i - \mu_i)^2} \quad (2)$$

где D - евклидово расстояние, x_i - значение i -го признака классифицируемого пикселя, μ_i - среднее значение i -го признака для данного класса, n - количество признаков.

Метод минимального расстояния часто используется в задачах анализа данных ДЗЗ благодаря своей простоте и высокой скорости обработки. Этот алгоритм особенно удобен при работе с большими массивами данных, поскольку он требует минимальных вычислительных ресурсов по сравнению с более сложными методами, такими как метод MLC или нейронные сети.

Однако главным недостатком метода является его чувствительность к распределению данных. Он предполагает, что классы имеют сферическую форму и одинаковый разброс признаков, что далеко не всегда соответствует реальным условиям. В случае, если классы имеют различные ковариационные структуры или данные внутри класса распределены неравномерно, метод минимального расстояния может приводить к неправильной классификации, особенно на границах классов.

Кроме того, метод не учитывает априорные вероятности классов и может быть менее точным в ситуациях, когда классы значительно различаются по численности. Например, если в исходных данных лесные массивы занимают гораздо большую площадь, чем водные объекты, метод минимального расстояния может ошибочно присваивать пиксели редких классов более крупным классам из-за их близости к центроидам доминирующих классов.

Для улучшения точности метод минимального расстояния может комбинироваться с другими алгоритмами, такими как метод расстояния Махаланобиса, который учитывает ковариационную структуру данных, или методы машинного обучения, способные адаптироваться к сложным распределениям данных. В современных геоинформационных системах (ГИС) метод минимального расстояния часто применяется как быстрая предварительная классификация, за которой следует уточняющая обработка с помощью более сложных алгоритмов, таких как градиентный бустинг, случайные леса или нейросетевые модели.

1.2.1 Инструкция по классификации методом MD

Требуется:

- QGIS – Quantum geoinformation system (рекомендуемая версия 3.10+)
- Плагин SCP (Semi-Automatic Classification Plugin) – бесплатный и мощный инструмент для обработки спутниковых снимков
- Спутниковое изображение (например, Landsat 8, Sentinel-2) в виде мультиканального растра

- ROI (Regions of Interest) - обучающие полигоны

Шаг - 1. Установка SCP в QGIS

1. Откройте QGIS.
2. Перейдите в Плагины → Управление и установка плагинов.
3. Найдите - Semi-Automatic Classification Plugin.
4. Нажмите - Установить и активируйте плагин.

После установки в правом верхнем углу появится панель инструментов SCP.

Шаг - 2. Загрузка и подготовка изображения

1. Перейдите в SCP → Band set (рисунок 1.12).
2. Загрузите каналы спутникового снимка (если они раздельны - объедините в Band set).
3. Убедитесь, что изображение корректно отобразилось и сохраните Band Set.

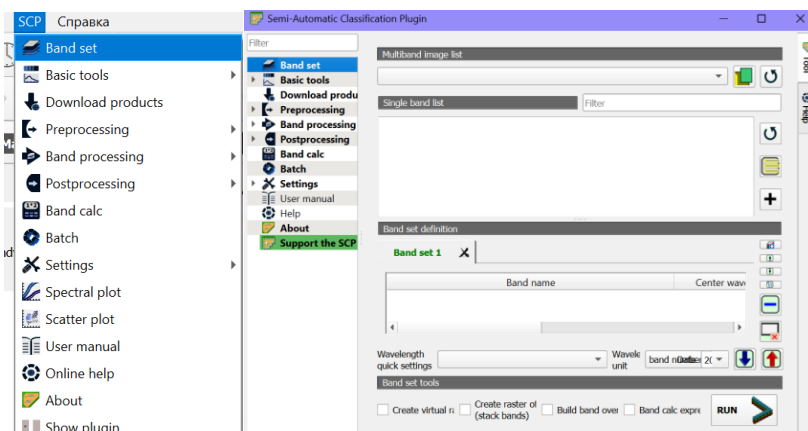


Рисунок 1.12 – Интерфейс модуля Semi-Automatic Classification Plugin

Шаг - 3. Создание обучающих полигонов (ROI)

1. В панели SCP откройте вкладку Training Input (рисунок 1.13).
2. Выберите Create ROI → выделите вручную полигоны для каждого класса (например, Cropland, Forest, Water и т.д.).
3. Назовите каждый класс и присвойте цвет.
4. Добавьте ROI в список обучающих данных (Вы должны создать по 5–10 ROI на класс для лучшей классификации)

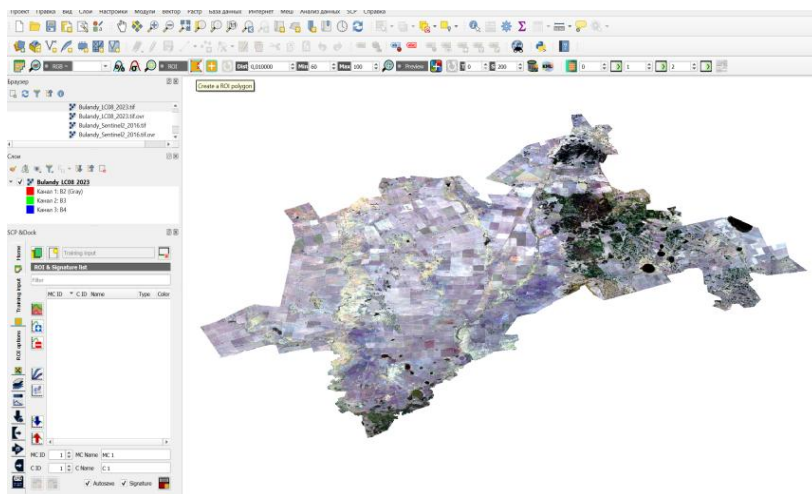


Рисунок 1.13 – Загруженный в QGIS спутниковый снимок

Шаг - 4. Классификация методом Minimum Distance

1. Перейдите в SCP → Band Processing → Classification → в разделе Classification Algorithm выберите Minimum Distance (рисунок 1.14).
2. Выберите ранее созданный Band set и ROI.
3. Установите порог угла (например, от 0.1 до 0.3 радиан). Чем меньше значение, тем строже классификация.
4. Укажите имя выходного файла (расширение .tif).
5. Нажмите Run.

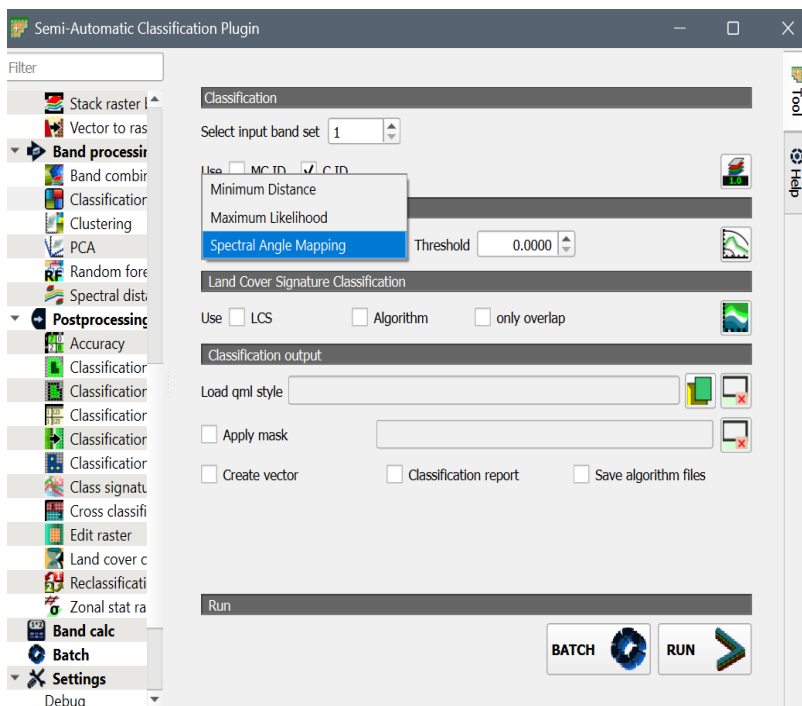


Рисунок 1.14 – Выбор модели классификации в модуле Semi-Automatic Classification Plugin

SCP выполнит анализ расстояний в многомерном спектральном пространстве от каждого пикселя до центроидов классов и присвоит пикселю ближайший по евклидову расстоянию класс.

Шаг - 5. Проверка и визуализация результатов

1. Классифицированный растр появится в окне карты (рисунок 1.15).

2. Используйте Identify Tool для просмотра присвоенных классов.

3. Чтобы визуализировать результат:

- Задайте уникальные цвета для каждого класса.
- Добавьте легенду и названия классов.

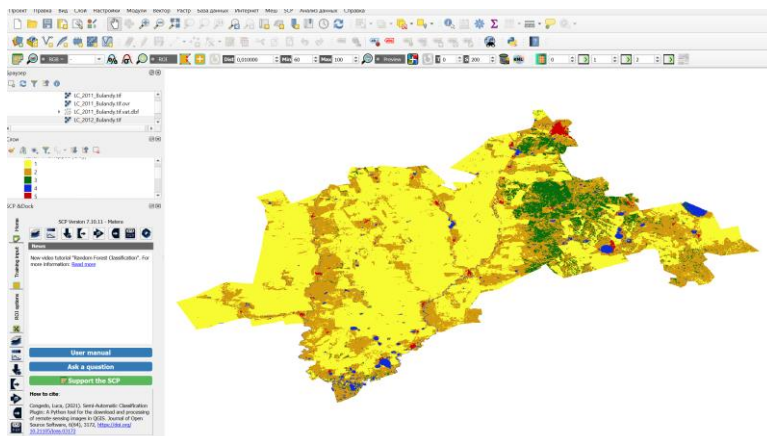


Рисунок 1.15 – Результат классификации методом MD

Шаг - 6. Оценка точности классификации

1. В SCP → Post Procesing → Accuracy → выберите Confusion Matrix (рисунок 1.16).

2. Укажите Ground Truth (ROI или проверочные точки).

3. Сравните результат классификации с валидационными данными.

4. Получите:

- Overall Accuracy
- Producer's Accuracy
- User's Accuracy
- Kappa statistics

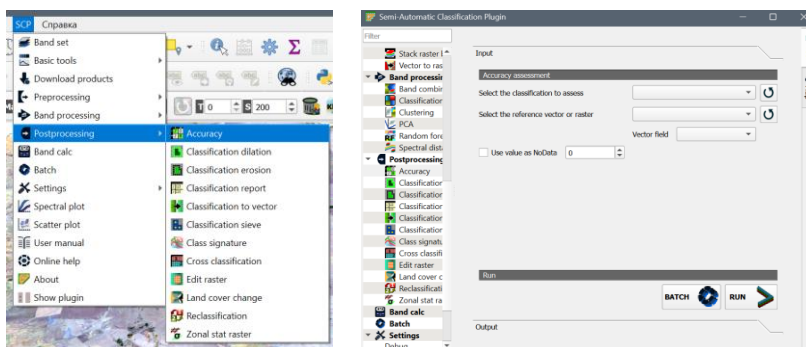


Рисунок 1.16 – Оценка точности в модуле Semi-Automatic Classification Plugin

1.3 Спектральный угловой картограф (Spectral Angle Mapper - SAM)

SAM – это метод классификации, который сравнивает спектральные сигнатуры пикселей с эталонными спектрами классов. SAM рассматривает спектры как векторы в n -мерном пространстве, где n – количество спектральных каналов. Метод вычисляет угол между вектором спектра пикселя и эталонными векторами классов. Пиксель относится к классу, с которым он образует наименьший угол.

На (рисунок 1.17) ниже представлен график, демонстрирующий метод спектрального угла. Этот метод используется для измерения угла между двумя спектральными векторами, что позволяет классифицировать объекты на основе их спектральных характеристик.

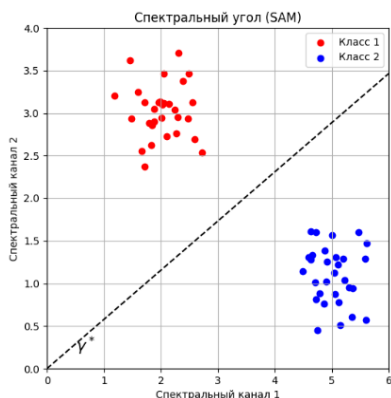


Рисунок 1.17 – График спектрального угла γ^* между двумя классами данных

График содержит следующие элементы: Два набора точек, представляющих два различных класса данных: точки красного цвета обозначают первый класс, синего цвета – второй класс. Оси координат представляют два спектральных канала: ось x – это первый спектральный канал, ось y – это второй спектральный канал. Линия, проведенная через начало координат, представляет собой угол γ^* , который является спектральным углом между

двумя классами данных. В данном случае угол составляет 30° (или $\frac{\pi}{6}$ радиан). Угол Y^* обозначен на графике в левом нижнем углу. Метод SAM измеряет угол между векторами классов в многомерном пространстве, что позволяет оценить их схожесть. Чем меньше угол между векторами, тем ближе классы друг к другу с точки зрения их спектральных характеристик. Математически угол между двумя векторами можно выразить как:

$$\alpha = \arccos \left(\frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \cdot \sqrt{\sum_{i=1}^n y_i^2}} \right) \quad (3)$$

где α - спектральный угол, x_i - значение i -го канала классифицируемого пикселя, y_i - значение i -го канала эталонного спектра, n - количество спектральных каналов.

Метод SAM широко применяется в гиперспектральном анализе для классификации объектов на основе их спектральных характеристик. В отличие от евклидова расстояния или методов статистического сравнения, SAM оценивает схожесть спектров, игнорируя различия в их общей интенсивности. Это делает его особенно полезным при анализе данных дистанционного зондирования, где освещение, атмосферные условия и сенсорные параметры могут влиять на абсолютные значения отражательной способности.

Одним из ключевых преимуществ метода является его устойчивость к изменениям освещенности. Например, если два объекта имеют одинаковую спектральную форму, но разную яркость из-за тени или угла падения солнечного света, SAM отнесет их к одному классу. Однако данный метод чувствителен к шуму в данных, особенно если количество спектральных каналов невелико. Кроме того, если спектры классов перекрываются, SAM может испытывать трудности с точной классификацией.

1.3.1 Инструкция по классификации методом SAM

Требуется:

- QGIS (рекомендуемая версия 3.10+)

- Плагин Semi-Automatic Classification Plugin (SCP) – бесплатный и мощный инструмент для обработки спутниковых снимков

- Спутниковое изображение (например, Landsat 8, Sentinel-2) в виде мультиканального растра

- ROI (Regions of Interest) - обучающие полигоны

Шаг - 1. Установка SCP в QGIS

1. Откройте QGIS.

2. Перейдите в Плагины → Управление и установка плагинов.

3. Найдите **Semi-Automatic Classification Plugin**.

4. Нажмите **Установить** и активируйте плагин.

После установки в правом верхнем углу появится панель инструментов SCP.

Шаг - 2. Загрузка и подготовка изображения

1. Перейдите в SCP → Band set (рисунок 1.18).

2. Загрузите каналы спутникового снимка (если они разделены - объедините в Band set).

3. Убедитесь, что изображение корректно отобразилось и сохраните **Band Set**.

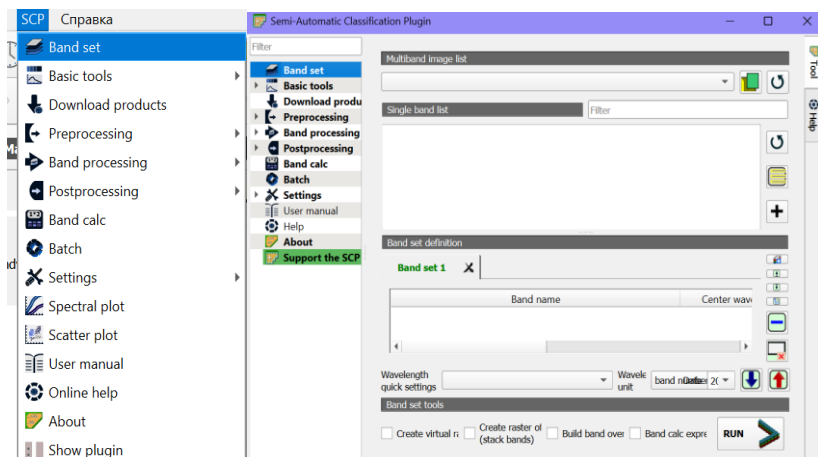


Рисунок 1.18 – Интерфейс модуля Semi-Automatic Classification Plugin

Шаг - 3. Создание обучающих полигонов (ROI)

1. В панели SCP откройте вкладку **Training Input** (рисунок 1.19).
2. Выберите Create ROI → выделите вручную полигоны для каждого класса (например, Cropland, Forest, Water и т.д.).
3. Назовите каждый класс и присвойте цвет.
4. Добавьте ROI в список обучающих данных (Вы должны создать по 5–10 ROI на класс для лучшей классификации)

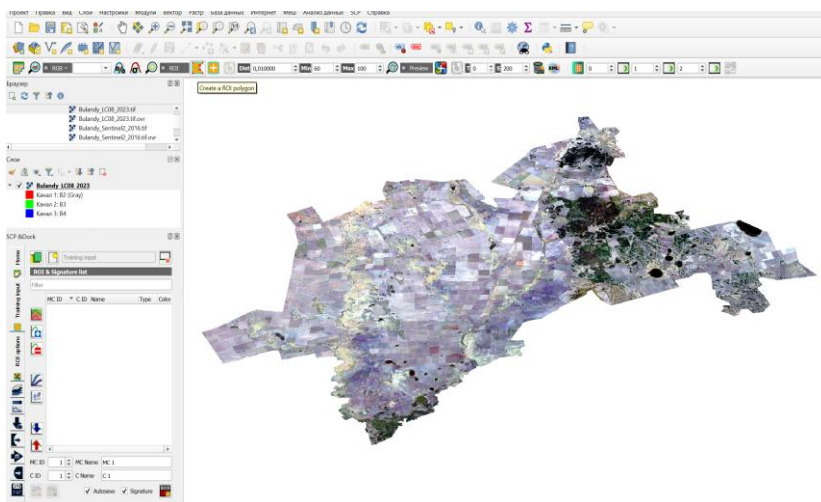


Рисунок 1.19 – Загруженный в QGIS спутниковый снимок

Шаг - 4. Классификация методом Spectral Angle Mapper (SAM)

1. Перейдите в SCP → Band Processing → Classification → в разделе **Classification Algorithm** выберите **Spectral Angle Mapper (SAM)** (рисунок 1.20).
2. Выберите ранее созданный Band set и ROI.
3. Установите порог угла (например, от 0.1 до 0.3 радиан). Чем меньше значение, тем строже классификация.
4. Укажите имя выходного файла (расширение .tif).
5. Нажмите **Run**.

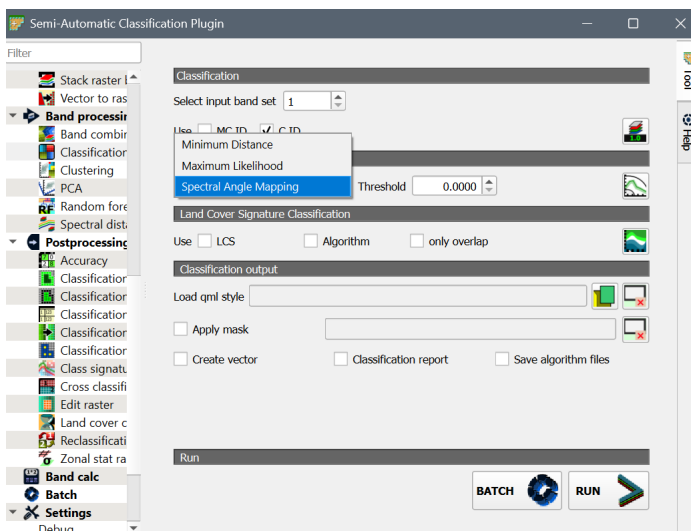


Рисунок 1.20 – Выбор модели классификации в модуле Semi-Automatic Classification Plugin

SAM вычислит угол между вектором спектрального сигнала пикселя и вектором среднего спектра каждого ROI-класса. Класс с наименьшим углом присваивается пикселю, если угол меньше установленного порога.

Шаг - 5. Проверка и визуализация результатов

1. Классифицированный растр появится в окне карты (рисунок 1.21).

2. Используйте Identify Tool для просмотра присвоенных классов.

3. Чтобы визуализировать результат:

- Задайте уникальные цвета для каждого класса.
- Добавьте легенду и названия классов.

Шаг - 6. Оценка точности классификации

1. В SCP → Post Procesing → Accuracy → выберите Confusion Matrix (рисунок 1.22).

2. Укажите **Ground Truth** (ROI или проверочные точки).

3. Сравните результат классификации с валидационными данными.

4. Получите:
 - **Overall Accuracy**
 - **Producer's Accuracy**
 - **User's Accuracy**
 - **Kappa statistics**

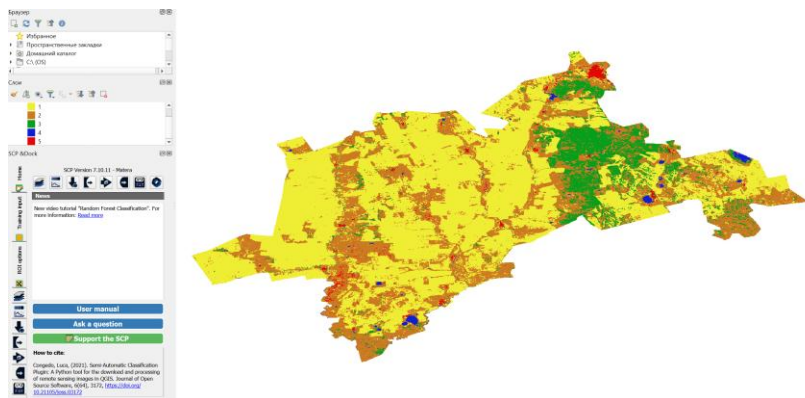


Рисунок 1.21 – Результат классификации методом Спектрального углового картографа

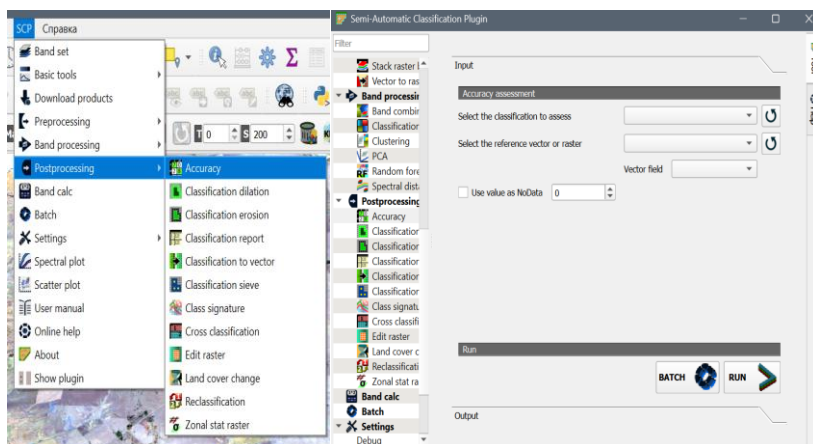


Рисунок 1.22 – Оценка точности в модуле Semi-Automatic Classification Plugin

Глава 2. КЛАССИФИКАЦИЯ С ИСПОЛЬЗОВАНИЕМ МАШИННОГО ОБУЧЕНИЯ (Machine Learning)

В этом разделе рассматриваются наиболее распространенные алгоритмы машинного обучения, применяемые для задач классификации LULC. К ним относятся: деревья классификации и регрессии (Classification and Regression Trees - CART), случайный лес (Random Forest - RF), метод опорных векторов (Support Vector Machine - SVM), метод K-средних, логистическая регрессия, бустинг и Байесовский классификатор.

2.1 Характеристика алгоритмов, основанных на Machine learning

CART – это мощный алгоритм машинного обучения, широко используемый в задачах классификации и прогнозирования. Метод основан на рекурсивном разбиении пространства признаков на основе минимизации определенной функции ошибки. Для классификации LULC этот метод особенно полезен, поскольку он адаптируется к сложным нелинейным зависимостям в данных и позволяет легко интерпретировать результаты. Основными особенностями CART являются:

- способность работать как с непрерывными, так и с категориальными признаками;
- устойчивости к выбросам и пропущенным значениям;
- интерпретируемости полученной модели.

Процесс построения дерева в CART основан на минимизации меры неоднородности (например, индекса Джини для классификации или среднеквадратичной ошибки для регрессии) в каждом узле:

$$Gini(t) = 1 - \sum_{j=1}^J p^2(j|t) \quad (7)$$

где $p(j|t)$ – доля наблюдений класса j в узле t , J - общее число классов. Для регрессионных задач используется критерий минимизации дисперсии:

$$MSE(t) = \sum_{i \in tN_t} \frac{1}{N_t} (y_i - \bar{y}_t)^2 \quad (8)$$

где N_t число наблюдений в узле t , y_i – значение целевой переменной для i -го наблюдения, $\bar{y}_t$ среднее значение целевой переменной в узле t .

RF – это ансамблевый метод, основанный на построении множества деревьев решений. Метод случайного леса строится на основе бэггинга (bagging) – техники ансамблевого обучения, при которой создается множество деревьев решений, каждое из которых обучается на случайной подвыборке обучающих данных. Итоговое предсказание получается путем усреднения результатов (для регрессии) или голосования (для классификации). Его преимущества для LULC классификации включают: высокую точность классификации; устойчивость к переобучению и способность работать с большим количеством признаков. Математически RF можно представить как:

$$\hat{f} = \frac{1}{B} \sum_{b=1}^B f_b(x) \quad (9)$$

где $\hat{f}$ – итоговый классификатор, B – число деревьев, $f_b(x)$ – отдельное дерево решений.

Метод широко используется в спутниковой картографии, мониторинге земного покрова, в сельском хозяйстве и городском планировании. Он позволяет анализировать сложные пространственные структуры данных и интегрировать различные источники информации (например, данные ДЗЗ, климатические показатели и топографические характеристики).

SVM – это алгоритм, который находит оптимальную гиперплоскость, разделяющую классы в многомерном пространстве признаков. Для LULC классификации SVM эффективен благодаря: способности работать с нелинейными границами классов; хорошей обобщающей способности и эффективности при высокой размерности данных.

В основе метода лежит поиск разделяющей гиперплоскости, которая отделяет классы таким образом, чтобы расстояние (margin) до ближайших точек каждого класса (опорных векторов) было

максимальным. Оптимальная гиперплоскость определяется следующим уравнением:

$$w^T x + b = 0$$

где: w – вектор весов, определяющий ориентацию гиперплоскости, x – вектор признаков, b – смещение (bias).

Метод широко используется для картирования земного покрова, мониторинга сельскохозяйственных угодий, анализа городских территорий и оценки деградации лесов. В частности, SVM эффективен при работе с мульти- и гиперспектральными снимками, где требуется обработка данных высокой размерности.

В сочетании с глубокими нейронными сетями (Deep Learning) метод SVM может использоваться на этапе финальной классификации, улучшая точность за счет структурированной интерпретации сложных признаков.

2.2 Инструкция по классификации LULC моделями Machine learning

В данной инструкции реализация моделей машинного обучения (CART, RF и SVM) выполняется на облачной платформе Google Earth Engine (GEE). Использование данной платформы позволяет эффективно обрабатывать большие объёмы спутниковых данных ДЗЗ и выполнять классификацию LULC с применением современных алгоритмов машинного обучения.

Следует отметить, что структура программного скрипта для реализации указанных алгоритмов является практически идентичной. Основные этапы обработки данных, включая подготовку входных спутниковых изображений, формирование обучающей выборки, извлечение спектральных признаков и выполнение классификации, реализуются с использованием единого программного шаблона. Различия между моделями заключаются лишь в выборе конкретного алгоритма машинного обучения, который применяется на этапе обучения классификатора.

Для выбора необходимой модели машинного обучения в скрипте предусмотрен отдельный этап (шаг 12), в котором пользователь должен вставить соответствующий фрагмент кода,

отвечающий за инициализацию выбранного алгоритма. В зависимости от целей исследования и характеристик анализируемых данных на данном этапе может быть активирован один из следующих классификаторов - CART, RF или SVM. Такой подход позволяет использовать единый алгоритмический каркас и при необходимости быстро переключаться между различными методами классификации без существенных изменений в структуре программы.

Представленная инструкция обеспечивает гибкость и воспроизводимость процесса классификации, что особенно важно при проведении сравнительного анализа различных алгоритмов машинного обучения при картографировании LULC.

2.2.1 Классификация LULC на платформе Google Earth Engine

1. Заходим на сайт GEE. [Ссылка: https://code.earthengine.google.com/](https://code.earthengine.google.com/)
2. Нам предстает интерфейс GEE (рисунок 2.1):

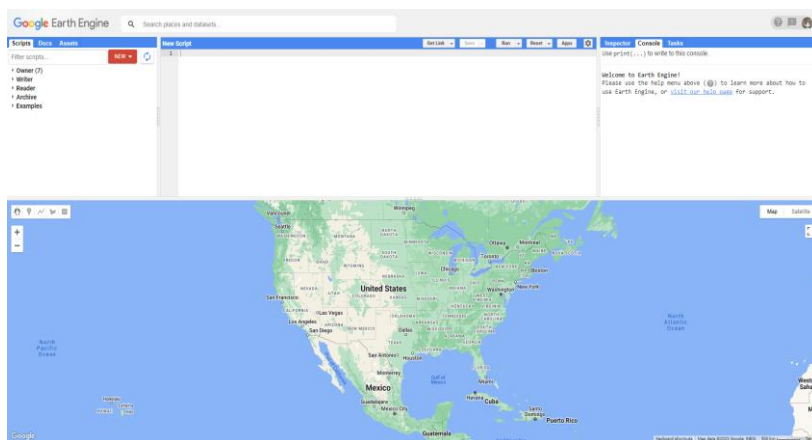


Рисунок 2.1 – Интерфейс GEE

3. Нам необходимо окно скрипта (рисунок 2.2):

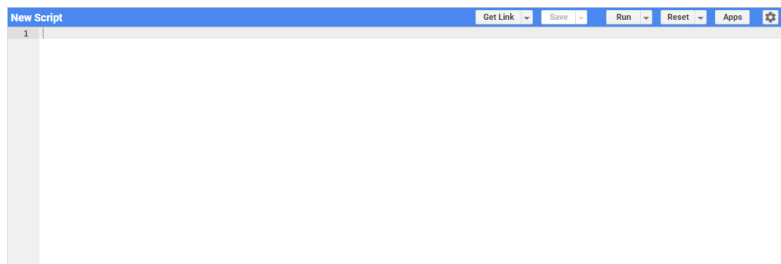


Рисунок 2.2 – Окно скрипта

4. Вставляем в окно скрипта данные строчки кода (рисунок 2.3):

```
// Landsat
var landsat =
ee.ImageCollection('LANDSAT/LC08/C02/T1_TOA')
    .filterDate('2016-04-01', '2016-09-01')
    .filter(ee.Filter.lt("CLOUD_COVER", 5))
    .select(['B2', 'B3', 'B4', 'B5'])
    .median().clip(table);
var visParams = {
  bands: ['B4','B3','B2'],
  min: 0,
  max: 0.3,
  gamma: 1,
};
Map.addLayer(landsat, visParams, 'True Color');
```

4.1. var – здесь обозначает Variables(переменные). Переменные могут хранить все виды информации, например строки, числа, функции или объекты, такие как растровые и векторные данные. var landsat – значит что данной переменной мы дали название landsat. После знака (=) идет функция под данной переменной.

4.1.1. ee.ImageCollection – код для создания Коллекции изображений. В данном случае это -'LANDSAT/LC08/C02/T1_TOA'.

Landsat – программа спутниковых снимков.

LC08 – спутник Ландсат 8.

C02 – 2-ая коллекция снимков.

T1_TOA – калиброванная отражательная способность верхней границы атмосферы (TOA).

Ссылка на источник:

https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LC08_C02_T1/

4.1.2. `filterDate('2016-04-01', '2016-09-01')` – функция выборки дат снимков.

4.1.3. `.filter(ee.Filter.lt("CLOUD_COVER", 5))` – функция выбора облачности снимков.

4.1.4. `.select(['B2', 'B3', 'B4', 'B5'])` – функция выбора каналов снимка. В данном случае B2 – Синий, B3 – Зеленый, B4 – Красный, B5 – Ближний инфракрасный

4.1.5. `.median()` – функция сокращения коллекции изображения, вычисляя среднюю между верхним и нижним показателем всех значений для каждого пикселя в огромном наборе всех совпадающих каналов.

4.1.6. `.clip(table)`; - функция обрезки снимков под наш полигон интересов.

4.2. `var visParams` – функция(переменная) которая отвечает за начала ветки функций для настройки визуализации слоя Landsat.

4.2.1. `bands: ['B4','B3','B2']`, - выбор каналов

`min: 0, max: 0.3`, - задача минимального и максимального значения в шкале канала.

`gamma: 1`, - Гамма-поправочный коэффициент.

4.2.2. `Map.addLayer(landsat, visParams, 'True Color')`; - создание слоя landsat. Landsat – тем самым мы отсылаемся на функцию для получения изображений со спутника Landsat. VisParams – нужно для установки ранее подготовленных параметров визуализации

5. Далее вставляем код для ночных изображений VIIRS (Visible Infrared Imaging Radiometer Suite). VIIRS - Комбинированные изображения среднемесячной яркости с использованием ночных данных из комплекта радиометров видимой инфракрасной визуализации (VIIRS) дневного/ночного диапазона. Ссылка на каталог VIIRS в Earth Engine Data Catalog: https://developers.google.com/earth-engine/datasets/catalog/NOAA_VIIRS_DNB_MONTHLY_V1_VCMCFG

Code Editor (JavaScript)

```
var dataset = ee.ImageCollection('NOAA/VIIRS/DNB/MONTHLY_V1/VCMSFG')
    .filter(ee.Filter.date('2017-05-01', '2017-05-31'));
var nighttime = dataset.select('avg_rad');
var nighttimeVis = {min: 0.0, max: 60.0};
Map.setCenter(-77.1056, 38.8904, 8);
Map.addLayer(nighttime, nighttimeVis, 'Nighttime');
```

Рисунок 2.3 – Вставленные в окно скрипта строки кода

В код внесем следующие изменения:

// VIIRS NLT

```
var noaaviirs = ee.ImageCollection('NOAA/VIIRS/DNB/MONTHLY_V1/VCMSL
CFG')
```

```
    .filter(ee.Filter.date('2022-01-01', '2022-12-31'))
    .select('avg_rad').median().clip(table)
```

```
var nighttimeVis = {min: 0, max: 10};
```

```
Map.addLayer(noaaviirs, nighttimeVis, 'VIIRS');
```

5.1. // VIIRS NLT – это примечание. Строчки кода начинающиеся с - // являются примечаниями.

5.2. var noaaviirs – функция, которой мы дали название спутникового изображения VIIRS.

5.3. .select('avg_rad') – выбор канала avg rad(среднее излучение)

5.4. .median() – функция сокращения коллекции изображения, вычисляя среднюю между верхним и нижним показателем всех значений для каждого пикселя в огромном наборе всех совпадающих каналов.

5.5. .clip(table); - функция обрезки снимков под наш полигон интересов.

5.6. Map.addLayer(noaaviirs, nighttimeVis, 'VIIRS'); - создание слоя VIIRS. noaaviirs – функция спутникового изображения VIIRS. nighttimeVis – параметры визуализации VIIRS. 'VIIRS' – название которое мы даем слою.

6. Далее вставляем код для расчета индексов: Ссылка: https://developers.google.com/earth-engine/tutorials/tutorial_api_06

```
// Indexes
// Compute the Normalized Difference Vegetation Index (NDVI).
var nir = landsat.select('B5');
var red = landsat.select('B4');
var ndvi = nir.subtract(red).divide(nir.add(red)).rename('NDVI');
var ndviParams = {min: -1, max: 1, palette: [ 'red', 'orange',
'yellow', 'green', '#006400']};
```

```
Map.addLayer(ndvi, ndviParams, 'NDVI');
```

6.1. var nir = landsat.select('B5'); - функция (ближнего инфракрасного) канала landsat.

6.1.1. landsat.select('B5') – выбор канала B5(ближний инфракрасный) из коллекции изображений landsat.

6.2. var red = landsat.select('B4'); - функция (красного) канала landsat.

6.2.1. landsat.select('B4') – выбор канала B4(красный) из коллекции изображений landsat.

```
6.3. var                                ndvi                                =
nir.subtract(red).divide(nir.add(red)).rename('NDVI');
```

- код для вычисления индекса ndvi с помощью каналов nir и red.

6.4. var ndviParams = {min: -1, max: 1, palette: ['red', 'orange', 'yellow', 'green', 'dark green']}; - параметры визуализации индекса 'NDVI'. Поскольку индекс 'NDVI' имеют шкалу от -1 до 1 мы напевую выставляем минимальное и максимальное значение(min: -1, max: 1,). Далее задаем слою цветовую шкалу от минимального к максимальному('red', 'orange', 'yellow', 'green', 'dark green'). Тем самым значение от -1 до 0,5 будет иметь цвет 'red', от 0,5 до 0 orange' и тд.

6.5. Map.addLayer(ndvi, ndviParams, 'NDVI'); - создание слоя 'NDVI'.

7. Вставляем в скрипт код для вычисления индекса 'NDWI': Ссылка: https://developers.google.com/earth-engine/tutorials/tutorial_api_06

```
// Compute the Normalized Difference Water Index ('NDWI').
```

```
var green = landsat.select('B3');
```

```
var                                ndwi                                =
nir.subtract(green).divide(nir.add(green)).rename('NDWI');
```

```
var ndwiParams = {min: -1, max: 1, palette: [ '#00008b', 'blue',
'yellow', 'orange', 'red']};
```

```
Map.addLayer(ndwi, ndwiParams, 'NDWI');
```

7.1. `var green = landsat.select('B3');` - функция (зеленого) канала `landsat`.

7.1.1. `landsat.select('B3')` – выбор канала B3 (зеленый) из коллекции изображений `landsat`.

7.2. `var ndwi = nir.subtract(green).divide(nir.add(green)).rename('NDWI');` - код для вычисления индекса `ndwi` с помощью каналов `nir` и `green`.

7.3. `var ndwiParams = {min: -1, max: 1, palette: [ 'Dark blue, 'blue', 'yellow', 'orange', 'red']};` - параметры визуализации индекса 'NDWI'. Поскольку индекс NDWI имеет шкалу от -1 до 1 мы напевую выставляем минимальное и максимальное значение (`min: -1, max: 1`). Далее задаем слою цветовую шкалу от минимального к максимальному ('Dark blue, 'blue', 'yellow', 'orange', 'red'). Тем самым значение от -1 до 0,5 будет иметь цвет "Dark blue", от 0,5 до 0 blue" и тд.

7.4. `Map.addLayer(ndwi, ndwiParams, 'NDWI');` - создание слоя 'NDWI'.

8. Далее добавляем код для объединения ночных изображений `noaaviirs` и индексов 'NDVI', 'NDWI'. Ссылка: <https://developers.google.com/earth-engine/apidocs/ee-image-addbands>

```
// AddBand to Landsat 8
```

```
var newBands = ee.Image([ noaaviirs, ndwi]);
```

```
var img = ndvi.addBands(newBands);
```

8.1. `var newBands = ee.Image([ noaaviirs, ndwi]);` - код для создания набора изображений `noaaviirs` и `ndwi`.

8.2. `var img = ndvi.addBands(newBands);` - добавление спутниковых изображений `noaaviirs`, `ndwi` и `ndvi` в набор спутниковых изображений – `img`.

9. Подготовка параметров набора обучающих данных для классификации. Также вставляем данный код в скрипт: Ссылка: <https://developers.google.com/earth-engine/apidocs/ee-classifier-train>

```
// Training dataset
```

```
var training = img.sample({
```

```
  region: table,
```

```
  scale: 30,
```

```
  numPixels: 1000
```

```
});
```

9.1. `var training = img.sample` – код для выборки коллекции изображений – `img` как объект классификации.

9.2. `region: table`, - выбор полигона `table` как зону интересов.

9.3. `scale: 30`, - разрешение в 30*30 метров

9.4. `numPixels: 1000` – количество пикселей на один образец обучающих данных.

10. Вставляем код экземпляра кластеризатора и его обучение. Ссылка: <https://developers.google.com/earth-engine/guides/clustering>

```
// Instantiate the clusterer and train it.
```

```
var clusterer = ee.Clusterer.wekaKMeans(22).train(training);
```

```
var result = img.cluster(clusterer);
```

```
var classes = result.randomVisualizer().toUint16();
```

11. Код для создания коллекции классов: Ссылка: <https://developers.google.com/earth-engine/apidocs/ee-featurecollection-classify>

```
var newfc = water
```

```
.merge(cropland)
```

```
.merge(grassland)
```

```
.merge(treecover)
```

```
.merge(builtup);
```

```
var clustbands = ['viz-red','viz-green','viz-blue'];
```

```
var label = 'lc';
```

```
var training = classes.select(clustbands).sampleRegions({
```

```
  collection: newfc,
```

```
  properties: [label],
```

```
  scale: 30
```

```
});
```

Данный код нужен для объединения классов в одну коллекцию и использования их как обучающей выборки для классификации.

11.1. `.merge` – функция слияния классов.

11.2. `var training = classes.select(clustbands).sampleRegions({`

```
  collection: newfc,
```

```
  properties: [label],
```

```
  scale: 30
```

```
});
```

 – данная часть кода отвечает за создание обучающей

выборки. `Collection` – выбор коллекции классов. `Properties` –

характеристики, в данном случае 'lc'. 'lc' – это характеристика классов, создание которых будет в конце методички.

2.2.2 Коды классификаторов машинного обучения

12. Необходимо вставить тот код классификатора что вам нужно использовать.

Код классификатора CART: Ссылка:

<https://developers.google.com/earth-engine/apidocs/ee-classifier-smilecart>

```
var trained = Classifier.smileCart
.train(training, label, clustbands);
var classified = classes.select(clustbands).classify(trained);
```

Код классификатора SVM: Ссылка:

<https://developers.google.com/earth-engine/apidocs/ee-classifier-libsvm>

```
var trained = ee.Classifier.libsvm({
  kernelType: 'RBF',
  gamma: 0.5,
  cost: 10
}).train(training, label, clustbands);
var classified = classes.select(clustbands).classify(trained);
```

Код классификатора RF: Ссылка:

<https://developers.google.com/earth-engine/apidocs/ee-classifier-smilerandomforest>

```
var trained = ee.Classifier.smileRandomForest(100)
.train(training, label, clustbands);
var classified = classes.select(clustbands).classify(trained);
```

12.1. var trained – сам классификатор RF.

12.2. var classified – функция для вычисления классификации на основе ранее созданных коллекции изображений и классов.

13. Код создания слоя классификации:

```
Map.addLayer(classified,
  {min: 1, max: 5, palette: ['yellow', 'orange',
    '#84fc03', 'blue', 'red']},
  'classification');
```

Map.centerObject(table,9)

13.1. Map.centerObject(table,9) – функция приближения к зоне интересов на мировой карте GEE.

14. Код для вычисления точности классификации: Ссылка: <https://developers.google.com/earth-engine/apidocs/ee-classifier-confusionmatrix>

```
print('Confusion matrix: ', trained.confusionMatrix());
print("Consumer's                accuracy"
      ,
trained.confusionMatrix().consumersAccuracy());
print("Producer's                accuracy"
      ,
trained.confusionMatrix().producersAccuracy());
print('Overall accuracy: ', trained.confusionMatrix().accuracy());
```

15. Код экспорта растра классификации на GoogleDrive:

```
Export.image.toDrive({
  image: classified,
  description: 'File_name',
  folder:'Folder',
  scale: 30,
  maxPixels: 1e10,
  region: table,
  crs:'EPSG:4326'
});
```

15.1. Export.image.toDrive – функция экспорта.

15.2. image: classified, - выбор выгружаемого изображения

15.3. description – название растра при выгрузке

15.4. folder – папка в которую будет экспортирован растр

15.5. scale – разрешение растра

15.6. maxPixels – максимальное количество пикселей в растре

15.7. region – территория экспортируемого растра

15.8. crs - Геодезическая система координат

16. Создание классов классификации.

16.1. В левой стороне интерфейса GEE имеются данные инструменты (рисунок 2.4):

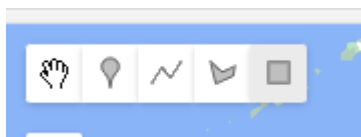


Рисунок 2.4 - Инструменты GEE

16.2. Нажимаем на прямоугольник и справа от этих инструментов появляется данная вкладка (рисунок 2.5):

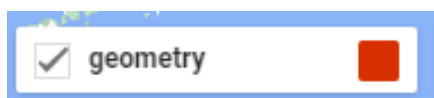


Рисунок 2.5 – Создание класса классификации

16.3. Наводим на нее мышь и ждем на шестеренку (Конфигурация) (рисунок 2.6):

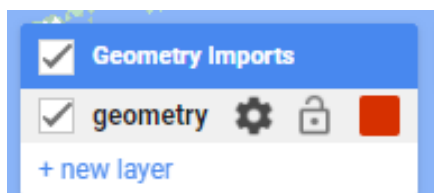


Рисунок 2.6 – Кнопка настройки класса

16.4. Появляется данное окно (рисунок 2.7):

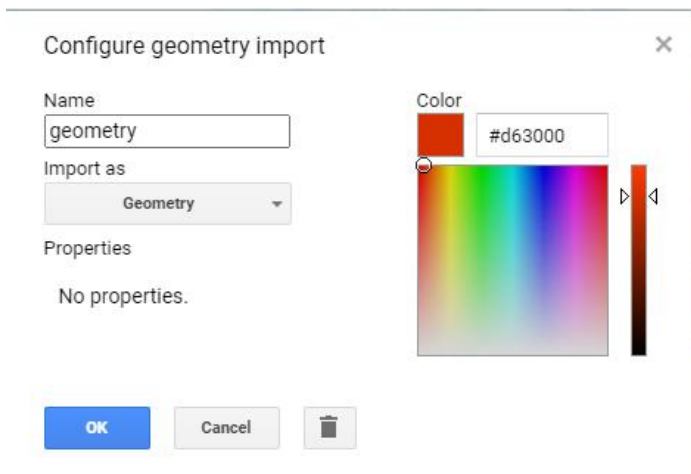


Рисунок 2.7 – Настройки класса

16.5. Во вкладку name – вводим название класса.

16.6. В Import as выбираем FeatureCollection (рисунок 2.8):

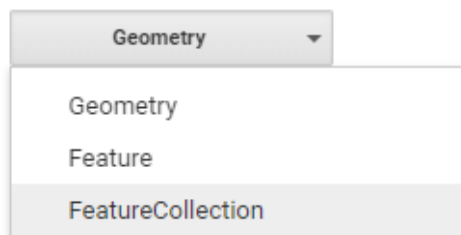


Рисунок 2.8 – Выбор типа объекта

16.7. Нажимаем +Property. Тем самым создаем Свойство класса (рисунок 2.9):



Рисунок 2.9 – Свойства класса

Задаем Property(Свойство) – lc и Value(Значение) от 1 до 5.
Для каждого класса нужно задать свое значение.

Cropland(Пашня) – значение - 1

Pasture(Пастбище) – значение - 2

Forest(Лес) – значение - 3

Water(Вода) – значение - 4

Builtup(Населенные пункты) – значение - 5

16.8. Как только задали свойство и значение нажимаем ОК.

17. После создания первого класса вновь наводим мышь на Geometry Imports и ждем + new layer и создаем все остальные классы (рисунок 2.10):

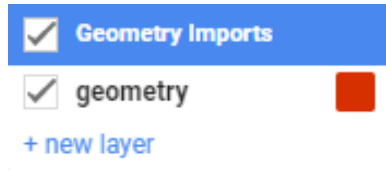


Рисунок 2.10 – Создание остальных классов

18. В конце мы получаем такую картину во вкладке Imports (рисунок 2.11):

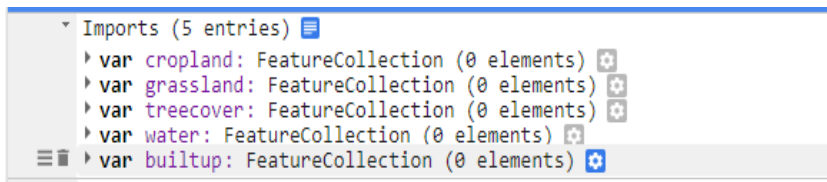


Рисунок 2.11 – Вкладка Imports

19. Теперь нам требуется шейп-файл для построения нашей зоны интереса. Для загрузки шейп-файла на левой части интерфейса GEE нажимаем на вкладку Assets (рисунок 2.12):

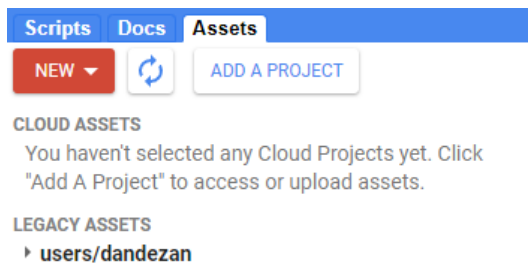


Рисунок 2.12 – Вкладка Assets

20. Нажимаем на кнопку NEW и выбираем Shape files (рисунок 2.13):

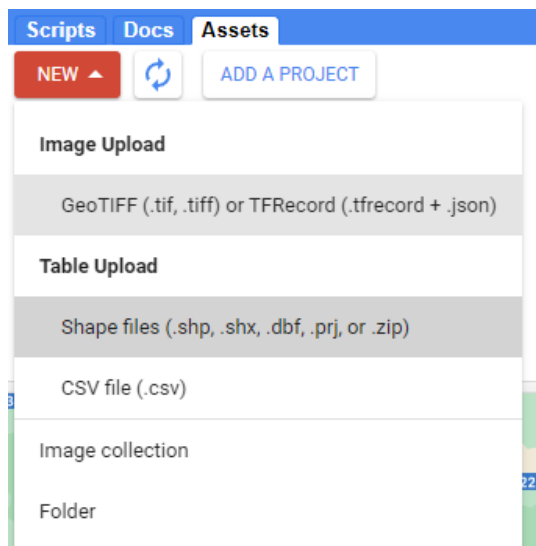


Рисунок 2.13 – Выбор формата загружаемого файла

21. В Source files жмем SELECT и выбираем наш шейп-файл. Выделяем все файлы помимо того что с окончанием .sbx (рисунок 2.14):

Имя	Дата изменения	Тип	Размер
Korgalzhyn.cpg	19.12.2022 10:30	Файл "CPG"	1 КБ
Korgalzhyn.dbf	19.12.2022 10:30	Файл "DBF"	1 КБ
Korgalzhyn.prj	19.12.2022 10:29	Файл "PRJ"	1 КБ
Korgalzhyn.sbn	19.12.2022 10:30	Файл "SBN"	1 КБ
Korgalzhyn.sbx	19.12.2022 10:30	Файл "SBX"	1 КБ
Korgalzhyn.shp	19.12.2022 10:30	Файл "SHP"	9 КБ
Korgalzhyn.shx	19.12.2022 10:30	Файл "SHX"	1 КБ

Рисунок 2.14 – Загрузка файла формата .shp.

22. В Asset Name задаем название файла и жмем UPLOAD(Загрузить) (рисунок 2.15):

Asset ID
users/dandezan/

Asset Name
Korgalzhyn

Properties

Metadata properties about the asset which can be edited during asset upload and after ingestion. The "system:time_start" property is used as the primary date of the asset.

Add start time
Add end time
Add property

Advanced options

Character encoding
UTF-8

?

Maximum error
1.0

?

☐ Split large geometries

?

[Learn more](#) about how uploaded files are processed.

CANCEL
UPLOAD

Рисунок 2.15 – Параметры загружаемого файла

23. После загрузки шейп-файла он вскоре появится во вкладке Assets (рисунок 2.16):

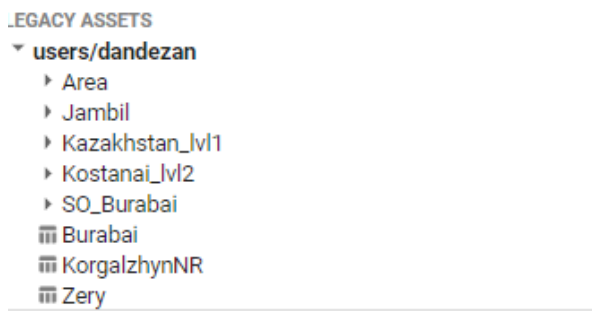


Рисунок 2.16 – Загруженный файл во вкладке Assets

24. Наводим на наш файл мышку и жмем на стрелочку. Тем самым мы загружаем шейп-файл в код.

25. Вскоре наш шейп-файл появляется в Imports под названием table (рисунок 2.17):

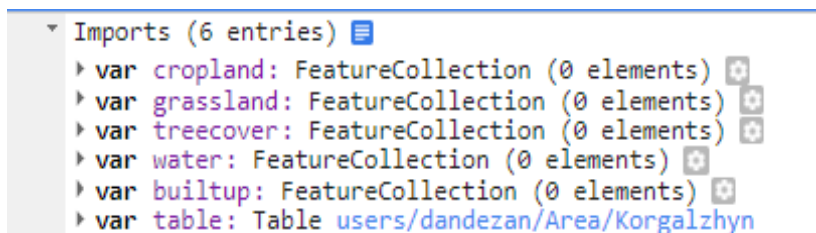


Рисунок 2.17 – Загруженный шейп файл во вкладке Imports

26. Теперь нажимаем кнопку Run, тем самым запускаем работу скрипта (рисунок 2.18):

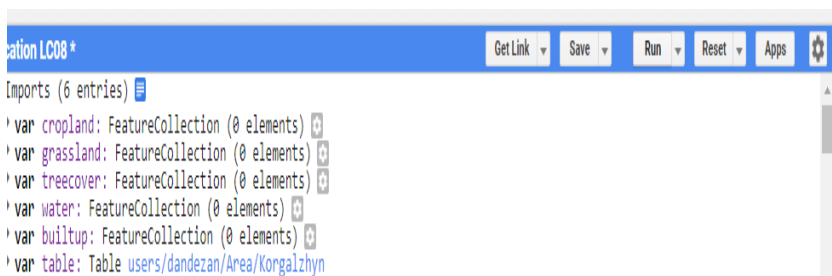


Рисунок 2.18 – Запуск скрипта

27. После запуска скрипта на экране начинают прогружаться ранее вычисленные индексы и слой RGB (True color) (рисунок 2.19):

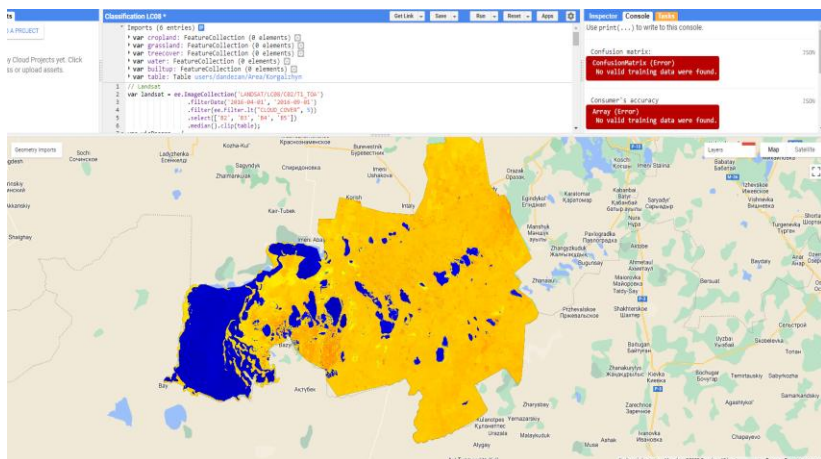


Рисунок 2.19 – Визуализация спутниковых снимков

Вы сразу заметите красные уведомления в вкладке Console на правой стороне интерфейса. Причиной их является в том, что хоть мы и создали классы, но мы не создали образцы классов.

28. Для этого наводим мышь на вкладку Layers в правой стороне интерфейса и убираем все галочки помимо True Color (рисунок 2.20):

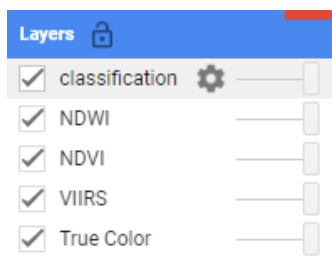


Рисунок 2.20 – вкладка Layers

29. Тогда мы видим такую картину (рисунок 2.21):

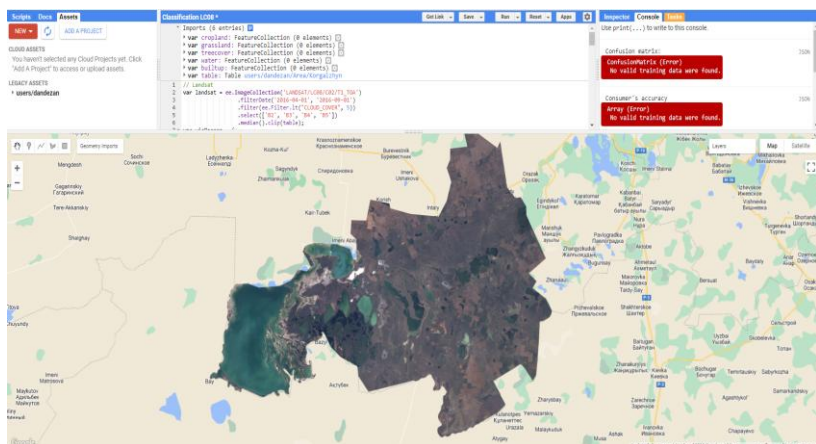


Рисунок 2.21 – Слой RGB (True Color)

30. Далее мы вновь наводим курсор на Geometry Imports и выбираем класс, а потом ждем на прямоугольник чуть левее (рисунок 2.22):

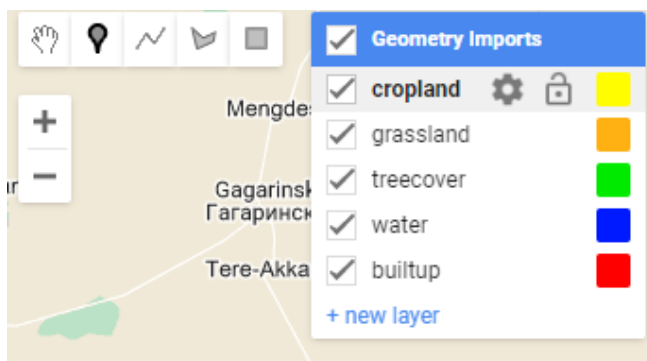


Рисунок 2.22 - Выбор одного из созданных классов

31. С помощью колесика мыши мы приближаем изображение и начинаем создавать образцы классов. Cropland к пашне, Pasture к пастбищам, Forest к лесу, Water к воде и Builtup к населенным пунктам (рисунок 2.23):

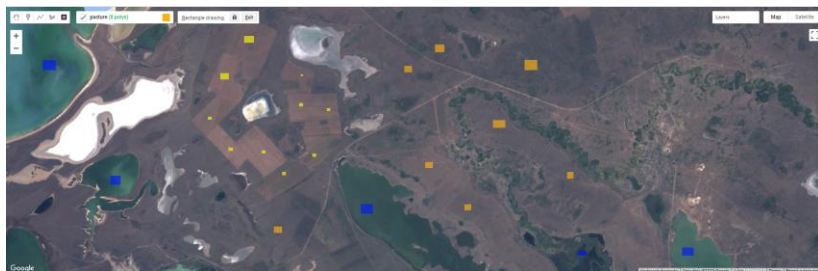


Рисунок 2.23 - Созданные на изображении образцы классов

32. Опционально сделать как можно больше образцов к каждому классу и после того, как зададите каждому классу образцы вновь нажимаем Run (рисунок 2.24):

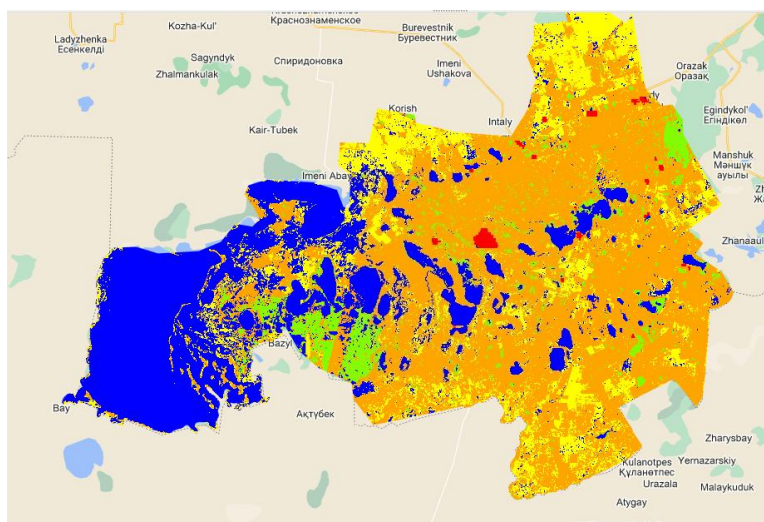


Рисунок 2.24 – Произведенная классификация

При условии, что классификация землепользования не сходится с реальной необходимо продолжать дополнять образцы в местах ошибочной классификации и повторять до тех пор, пока классификация не станет достоверной.

33. Для того чтобы выгрузить растр классификации на правой части интерфейса нажимаем на вкладку Tasks и видим UNSUBMITTED TASKS. Там будет ваш файл с заданным вами названием. Нажимаем Run (рисунок 2.25):

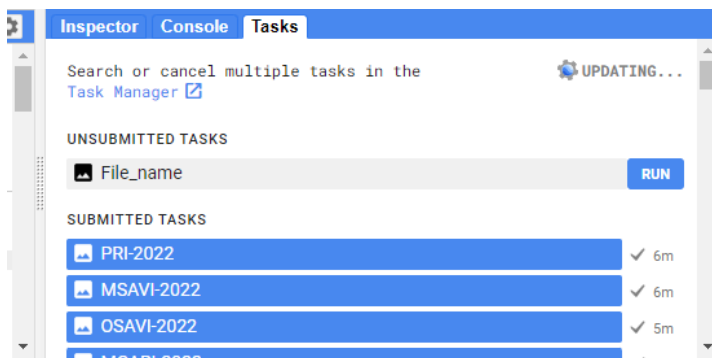


Рисунок 2.25 – Вкладка Tasks

34. После нажатия Run появляется данное окно (рисунок 2.26):

Task: Initiate image export

Task name (no spaces) *

File_name

Coordinate Reference System (CRS)

EPSG:4326

Scale (m/px)

30

DRIVE
CLOUD STORAGE
EE ASSET

Drive folder

Folder

Filename *

File_name

File format *

GEO_TIFF

CANCEL
RUN

Рисунок 2.26 – Вкладка загрузки tif файла классификации

B Task name задается название задачи загрузки
 Coordinate Reference System (CRS) - Геодезическая система
 Scale – разрешение раstra
 Drive folder – папка в GoogleDrive в которую будет выгружен растр

Filename – название с которым файл будет выгружен в GoogleDrive

File format – файл формата. В данном случае нам нужен растр, потому что выгружен будет в формате GEO_TIFF

Вновь жмем RUN и наш растр начнет обработку и спустя 5-9 минут будет выгружен на GoogleDrive.

Ход экспорта раstra можно наблюдать в той же вкладке Tasks (рисунок 2.27):

SUBMITTED TASKS

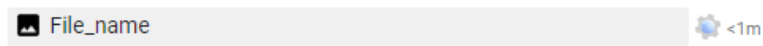


Рисунок 2.27 – Загружаемый файл во вкладке Tasks

```

    Полный скрипт для классификации на платформе GEE
    // Landsat8
    var                                landsat                                =
    ee.ImageCollection('LANDSAT/LC08/C02/T1_TOA')
        .filterDate('2023-04-01', '2023-09-01')
        .filter(ee.Filter.lt("CLOUD_COVER", 10))
        .select(['B2', 'B3', 'B4', 'B5'])
        .median().clip(table);
    var visParams = {
        bands: ['B4','B3','B2'],
        min: 0,
        max: 0.3,
        gamma: 1,
    };
    Map.addLayer(landsat, visParams, 'LC08');
    // VIIRS NLT
    var                                noaaviirs                                =
    ee.ImageCollection('NOAA/VIIRS/DNB/MONTHLY_V1/VCMSL
    CFG')
  
```

```

        .filter(ee.Filter.date('2017-01-01', '2017-12-31'))
        .select('avg_rad').median().clip(table)
var nighttimeVis = {min: 0, max: 50};
Map.addLayer(noaaviirs, nighttimeVis, 'VIIRS');
// DMSP-OLS NLT
var dmspols = ee.ImageCollection('NOAA/DMSP-
OLS/NIGHTTIME_LIGHTS')
        .filter(ee.Filter.date('2012-01-01', '2012-12-31'))
        .select('stable_lights').median().clip(table);
var dmspolsVis = {
    min: 6.0,
    max: 60.0,
};
Map.addLayer(dmspols, dmspolsVis, 'DMSP-OLS');
// Indexes
// Compute the Normalized Difference Vegetation Index (NDVI).
var nir = landsat.select('B5');
var red = landsat.select('B4');
var ndvi = nir.subtract(red).divide(nir.add(red)).rename('NDVI');
var ndviParams = {min: -1, max: 1, palette: [ 'red', 'orange',
'yellow', 'green', '#006400']};
Map.addLayer(ndvi, ndviParams, 'NDVI');
// Compute the Normalized Difference Water Index (NDWI).
var green = landsat.select('B3');
var ndwi = nir.subtract(green).divide(nir.add(green)).rename('NDWI');
var ndwiParams = {min: -1, max: 1, palette: [ '#00008b', 'blue',
'yellow', 'orange', 'red']};
Map.addLayer(ndwi, ndwiParams, 'NDWI');
// AddBand to Landsat 8
var newBands = ee.Image([ noaaviirs, ndwi, ndvi]);
var img = ndvi.addBands(newBands);
// Training dataset
var training = img.sample({
    region: table,
    scale: 30,
    numPixels: 1000
});

```

```

// Instantiate the clusterer and train it.
var clusterer = ee.Clusterer.wekaKMeans(22).train(training);
var result = img.cluster(clusterer);
var classes = result.randomVisualizer().toUint16();
var newfc = water
.merge(cropland)
.merge(grassland)
.merge(treecover)
.merge(builtup);
var withRandom = newfc.randomColumn('random');
var split = 0.3;
var trainingFc = withRandom.filter(ee.Filter.lt('random', split));
var validationFc = withRandom.filter(ee.Filter.gte('random',
split));
var clustbands = ['viz-red','viz-green','viz-blue'];
var label = 'lc';
var training = classes.select(clustbands).sampleRegions({
  collection: validationFc,
  properties: [label],
  scale: 30
});
var trained = ee.Classifier.smileCart()
.train(training, label, clustbands);
var classified = classes.select(clustbands).classify(trained);
var validated = training.classify(trained);
Map.addLayer(classified,
  {min: 1, max: 5, palette: ['yellow', 'orange',
    '#009900','blue','red']},
  'classification');
Map.centerObject(table,9)
print('Confusion matrix (validation): ',
validated.errorMatrix(label, 'classification'));
print("Consumer's accuracy (validation)" ,
validated.errorMatrix(label, 'classification').consumersAccuracy());
print("Producer's accuracy (validation)" ,
validated.errorMatrix(label, 'classification').producersAccuracy());
print('Overall accuracy (validation): ',
validated.errorMatrix(label, 'classification').accuracy());

```

```

print("Kappa statistic", trained.confusionMatrix().kappa());
Export.image.toDrive({
    image: landsat,
    description: 'Bulandy_LC08_2023',
    folder:'Bulandy-Indexes',
    scale: 30,
    maxPixels: 1e11,
    region: table,
    crs:'EPSG:4326'
});
Export.image.toDrive({
    image: noaaviirs,
    description: 'Bulandy_VIIRS_2017',
    folder:'Bulandy-Indexes',
    scale: 30,
    maxPixels: 1e11,
    region: table,
    crs:'EPSG:4326'
});
Export.image.toDrive({
    image: dmspols,
    description: 'Bulandy_DMSP-OLS_2012',
    folder:'Bulandy-Indexes',
    scale: 30,
    maxPixels: 1e11,
    region: table,
    crs:'EPSG:4326'
});

```

2.3 Ограничения алгоритмов машинного обучения

Можно выделить следующие общие ограничения методов машинного обучения для классификации LULC:

- 1) Склонность к переобучению, особенно на небольших выборках данных.
- 2) Нестабильность результатов при небольших изменениях в обучающих данных.
- 3) Сложность интерпретации некоторых моделей.

- 4) Повышенные требования к вычислительным ресурсам для сложных моделей.
- 5) Чувствительность к выбору гиперпараметров и настройке модели.
- 6) Потенциально низкая производительность на больших наборах данных.
- 7) Необходимость предварительного определения числа классов или кластеров.
- 8) Чувствительность к выбору начальных условий.
- 9) Тенденция к формированию кластеров определенной формы.
- 10) Предположение о линейности связей между признаками и целевой переменной.
- 11) Чувствительность к мультиколлинеарности признаков.
- 12) Чувствительность к шуму в данных.
- 13) Предположение о независимости признаков в некоторых моделях.
- 14) Чувствительность к качеству оценки вероятностных характеристик.

Глава 3. КЛАССИФИКАЦИЯ С ИСПОЛЬЗОВАНИЕМ ГЛУБОКОГО ОБУЧЕНИЯ (Deep Learning - DL)

Deep learning и искусственный интеллект на платформах облачных вычислений открывают новые возможности в классификации LULC, позволяя автоматически извлекать признаки и достигать высокой точности классификации. На изображении представлена иерархическая структура алгоритмов DL в виде графа (рисунок 3.1).

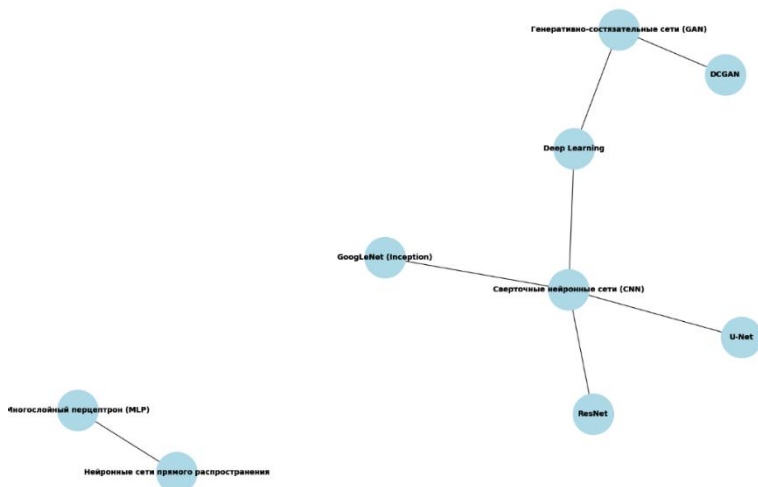


Рисунок 3.1 - Иерархическая структура алгоритмов глубокого обучения

Наиболее часто используемые архитектуры глубокого обучения для классификации LULC включают:

- искусственные нейронные сети (Artificial Neural Network - ANN): вычислительные системы или машины, созданные для моделирования аналитических действий, совершаемых человеческим мозгом;
- многослойный перцептрон (Multilayer perceptron - MLP): базовая архитектура нейронной сети, состоящая из входного слоя, одного или нескольких скрытых слоев и выходного слоя. Эффективен для моделирования сложных нелинейных зависимостей в данных.

- сверточные нейронные сети (Convolutional Neural Network-CNN): эффективны для извлечения пространственных признаков из изображений.

Далее рассмотрим использование нейронных сетей для классификации LULC на примере CNN.

3.1 Сверточные нейронные сети (Convolutional Neural Network - CNN)

CNN – это специализированная архитектура нейронных сетей, разработанная для эффективной обработки данных с сеточной топологией, таких как изображения. CNN особенно эффективны для задач классификации LULC благодаря: способности автоматически извлекать пространственные признаки; инвариантности к сдвигам и искажениям входных данных и эффективному использованию параметров за счет разделения весов. Основные компоненты CNN включают сверточные слои, слои подвыборки (пулинга) и полносвязные слои. Операция свертки может быть представлена как:

$$(f \cdot g)(x, y) = \sum_{i=-a}^a \sum_{j=-b}^b f(i, j) g(x - i, y - j) \quad (16)$$

где f – входное изображение, g – ядро свертки, a и b – размеры ядра.

CNN эффективно обрабатывают многоканальные спутниковые и аэрофотоснимки, автоматически выявляя текстурные и спектральные закономерности, что особенно важно для классификации сложных ландшафтов и объектов земного покрова. За счет многослойной структуры сети способны иерархически извлекать признаки – от низкоуровневых (границы, текстуры) до высокоуровневых (типы землепользования). Использование пулинга (максимального или усредняющего) снижает размерность данных, улучшая обобщающую способность модели. В сочетании с полносвязными слоями CNN позволяют достичь высокой точности классификации LULC, минимизируя потребность в ручной инженерии признаков.

3.3.1 Инструкция по классификации LULC с использованием CNN

Шаг - 1. ПОДГОТОВКА ПЛАТФОРМЫ

Шаг - 1.1. Установка Python

Убедитесь, что у вас установлен Python 3.12 или новее. Для проверки версии Python введите строчку ниже в командную строку PowerShell:

Проверка версии Python:

```
python --version
```

Если Python не установлен, скачайте его с официального сайта: <https://www.python.org/downloads/>

Шаг - 1.2. Установка зависимостей

Создайте файл формата txt с названием requirements в папке что будет использоваться для хранения всех скриптов и данных для дальнейшей работы. В созданный файл введите строки ниже:

```
# Deep Learning framework
```

```
torch>=2.0.0
```

```
# Data processing
```

```
numpy>=1.24.0
```

```
pandas>=2.0.0
```

```
geopandas>=1.1.2
```

```
# Geospatial data processing
```

```
rasterio>=1.3.0
```

```
# Machine Learning utilities
```

```
scikit-learn>=1.3.0
```

```
# Progress bars
```

```
tqdm>=4.65.0
```

Далее откройте командную строку PowerShell (терминал) и откройте путь к папке, где сохранен файл txt.

```
cd <ПУТЬ_К_ПАПКЕ_CNN>
```

Пример:

Путь к папке: C:\Users\Desktop\DL\CNN

B	PowerShell:	PS	C:\Users>	cd
C:\Users\OneDrive\Desktop\DL\CNN				

После открытия пути к папке проекта, выполните:

```
pip install -r requirements.txt
```

Это установит все необходимые библиотеки:

- PyTorch (для нейронных сетей)
- Rasterio (для работы с GeoTIFF)
- Pandas (для работы с CSV)
- NumPy (для вычислений)
- scikit-learn (для метрик)
- tqdm (для прогресс-баров)

Шаг - 2. ФОРМИРОВАНИЕ ИСХОДНЫХ ДАННЫХ

Шаг - 2.1. Структура папок

Важная информация о путях

Внимание: В данной методичке используются условные обозначения путей вида ``<ПУТЬ_К_ПАПКЕ_CNN>`'. Вы должны заменить эти обозначения на реальные пути к папкам на вашем компьютере.

Примеры путей:

- Если папка на рабочем столе:

``C:\Users\ВашеИмя\Desktop\DL method\CNN``

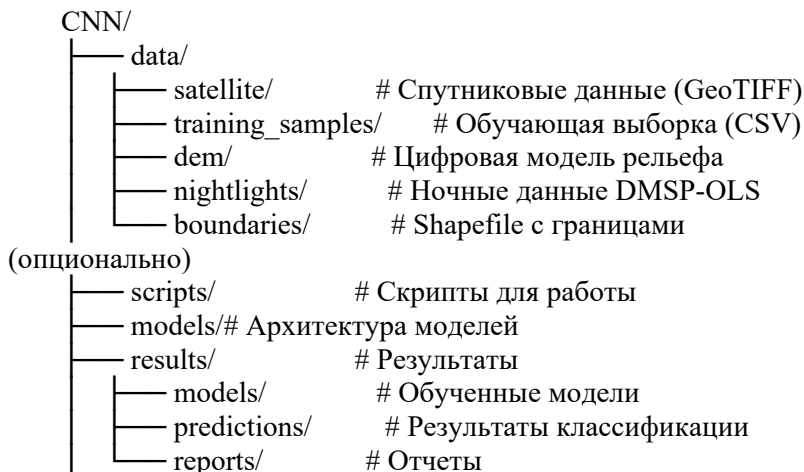
- Если папка в другом месте: ``D:\Projects\DL method\CNN``

- Если папка в документах:

``C:\Users\ВашеИмя\Documents\DL method\CNN``

Важно: Путь должен указывать именно на папку `CNN`, внутри которой будут находиться данные для работы классификации.

В папке под CNN создайте дополнительные папки по структуре ниже:



Шаг - 2.2. Требования к исходным данным

Спутниковые данные (обязательно)

Спутниковые снимки для классификации сохраняйте в папке data/ satellite что мы ранее создали:

Требования:

- Формат: GeoTIFF (.tif)
- Минимум: 4 канала (Blue, Green, Red, NIR)
- Рекомендуется: 6 каналов (Blue, Green, Red, NIR, SWIR1, SWIR2)
- Система координат: Любая (будет использована для всех остальных данных)

- Разрешение: Любое (рекомендуется 30м или выше)

Обучающая выборка (обязательно)

Создайте файл в формате csv - training_samples, по пути data/training_samples/

Требования:

- Колонки: `class`, `x`, `y` (обязательно) (рисунок 3.2)
- `class`: номер класса (1-5)
 - 1 = Пашня
 - 2 = Пастбища
 - 3 = Леса
 - 4 = Вода
 - 5 = Застроенные территории
- `x`, `y`: координаты в системе координат спутниковых данных (WGS84 или UTM)
- Минимум: 500 точек на класс (всего 2500 точек)
- Рекомендуется: 2500 точек на класс (всего 12500 точек)

	class	x	y
	1	70,088939	52,521665
	1	69,555879	52,373173
	1	70,526868	52,336792
	1	70,086244	52,155961
	1	70,156582	52,506843
	1	70,892302	52,332749
	1	70,015097	52,185875
	1	69,723774	52,062177
	1	70,409368	52,537026

Рисунок 3.2 - Пример таблицы классов

Создание выборки:

1. Откройте спутниковое изображение в QGIS или ArcGIS
2. Создайте точечный слой (shapefile)
3. Добавьте точки для каждого класса
4. Экспортируйте в CSV с колонками `class`, `x`, `y`

ДЕМ - Цифровая модель рельефа (рекомендуется)

Цифровая модель рельефа для классификации сохраняйте в папке data/dem.

Требования:

- Формат: GeoTIFF (.tif)
- Одноканальный растр высот
- Источник: SRTM, ASTER GDEM, или другие
- Автоматически ресамплируется к разрешению

спутниковых данных

Загрузка данных:

- SRTM: <https://earthexplorer.usgs.gov/>
- ASTER GDEM: <https://search.earthdata.nasa.gov/>

Ночные данные DMSP-OLS (рекомендуется)

Ночные данные для классификации сохраняйте в папке data/nightlights.

Требования:

- Формат: GeoTIFF (.tif)
- Одноканальный растр ночной подсветки
- Источник: DMSP-OLS или VIIRS
- Помогает идентифицировать застроенные территории

Загрузка данных:

- DMSP-OLS: <https://eogdata.mines.edu/products/vnl/>
- VIIRS: <https://eogdata.mines.edu/products/vnl/>

Границы области (рекомендуется)

Файл: Shapefile с границами области исследования

Требования:

- Формат: ESRI Shapefile (.shp, .shx, .dbf)
- Тип геометрии: Полигон (Polygon)
- Система координат: Та же, что у спутниковых данных

Шаг - 3. РАСШИРЕНИЕ ОБУЧАЮЩЕЙ ВЫБОРКИ

Шаг - 3.1. Зачем расширять выборку?

Расширение выборки увеличивает количество обучающих примеров, что обычно приводит к повышению точности модели на 5-10%.

Шаг - 3.2. Как работает расширение

Скрипт создает новые точки вокруг существующих:

- Для каждой исходной точки создается несколько новых точек

- Новые точки находятся в радиусе 3-5 пикселей от исходной

- Все новые точки проверяются на принадлежность границам области

Шаг - 3.3 Создание скрипта

Создайте файл формата .txt в папке scripts и назовите его 00_expand_training_samples. Далее откройте файл через «Блокнот» и вставьте скрипт:

```
"""Скрипт для расширения обучающей выборки
(увеличение количества точек в 5 раз)"""
import sys
sys.stdout.reconfigure(encoding='utf-8')
import os
import pandas as pd
import numpy as np
from pathlib import Path
import rasterio
from rasterio.transform import xy
import geopandas as gpd
from shapely.geometry import Point
# Добавляем родительскую директорию в путь для импорта
parent_dir =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
sys.path.insert(0, parent_dir)
import config
def expand_training_samples(input_csv, output_csv,
expansion_factor=5, max_neighbors=3):
    """
```

Расширяет обучающую выборку, создавая новые точки вокруг существующих

Args:

input_csv: путь к исходному CSV файлу с выборкой

```

output_csv: путь к выходному CSV файлу
expansion_factor: во сколько раз увеличить выборку (по
умолчанию 5)
max_neighbors: максимальное расстояние в пикселях для
создания новых точек
"""
print("=" * 70)
print("РАСШИРЕНИЕ ОБУЧАЮЩЕЙ ВЫБОРКИ")
print("=" * 70)
print()
# Загружаем исходную выборку
print(f"Загрузка исходной выборки из {input_csv}...")
df = pd.read_csv(input_csv)
print(f" [ОК] Загружено {len(df):,} точек")
# Проверяем распределение по классам
class_counts = df['class'].value_counts().sort_index()
print("\nРаспределение по классам (исходное):")
for cls, count in class_counts.items():
    print(f" Класс {cls}: {count:,} точек")
print()
# Загружаем спутниковые данные для проверки валидности
координат
satellite_path = Path(config.get_satellite_path())
print(f"Загрузка спутниковых данных для проверки
координат...")
with rasterio.open(satellite_path) as src:
    height, width = src.height, src.width
    transform = src.transform
    crs = src.crs
    print(f" [ОК] Размер изображения: {height} x {width}")
    print()
    # Загружаем границы Буланды (если указаны)
    boundaries_gdf = None
    if config.BOUNDARIES_FILE:
        print(f"Загрузка границ Буланды:
{config.BOUNDARIES_FILE}...")
        boundaries_gdf = gpd.read_file(config.BOUNDARIES_FILE)
        if boundaries_gdf.crs != crs:

```

```

boundaries_gdf = boundaries_gdf.to_crs(crs)
print(f" [OK] Загружено {len(boundaries_gdf)} полигонов")
print()
# Создаем расширенную выборку
expanded_samples = []
print("Создание новых точек...")
for class_id in sorted(df['class'].unique()):
    class_df = df[df['class'] == class_id].copy()
    num_original = len(class_df)
    num_target = num_original * expansion_factor
    num_new = num_target - num_original
    print(f"\nКласс {class_id}: {num_original:,} →
{num_target:,} точек (+{num_new:,})")
    # Добавляем исходные точки
    expanded_samples.append(class_df.copy())
    # Создаем новые точки вокруг существующих
    new_points = []
    attempts = 0
    max_attempts = num_new * 10 # Максимум попыток
    while len(new_points) < num_new and attempts <
max_attempts:
        attempts += 1
        # Выбираем случайную исходную точку
        base_idx = np.random.randint(0, len(class_df))
        base_row = class_df.iloc[base_idx]
        # Создаем новую точку в окрестности (случайное
смещение)
        offset_row = np.random.randint(-max_neighbors,
max_neighbors + 1)
        offset_col = np.random.randint(-max_neighbors,
max_neighbors + 1)
        new_row = base_row['row'] + offset_row
        new_col = base_row['col'] + offset_col
        # Проверяем, что координаты в пределах изображения
        if 0 <= new_row < height and 0 <= new_col < width:
            # Преобразуем в географические координаты
            x, y = xy(transform, new_row, new_col)

```

```

        # Проверяем, что точка внутри границ (если границы
указаны)
        if boundaries_gdf is not None:
            point = Point(x, y)
            if not boundaries_gdf.geometry.contains(point).any():
                continue
        # Проверяем, что такая точка еще не добавлена
        is_duplicate = False
        for existing in new_points:
            if abs(existing['row'] - new_row) < 2 and abs(existing['col'] -
new_col) < 2:
                is_duplicate = True
                break
            if not is_duplicate:
                new_points.append({
                    'class': class_id,
                    'x': x,
                    'y': y,
                    'row': new_row,
                    'col': new_col
                })
        if len(new_points) > 0:
            new_df = pd.DataFrame(new_points)
            expanded_samples.append(new_df)
            print(f" [OK] Создано {len(new_points):,} новых точек")
        else:
            print(f" [WARNING] Не удалось создать новые точки для
класса {class_id}")
        # Объединяем все точки
        print("\nОбъединение всех точек...")
        expanded_df = pd.concat(expanded_samples,
ignore_index=True)
        # Удаляем дубликаты (если есть)
        expanded_df = expanded_df.drop_duplicates(subset=['row',
'col'], keep='first')
        # Перемешиваем
        expanded_df = expanded_df.sample(frac=1,
random_state=42).reset_index(drop=True)

```

```

print(f" [OK] Итого: {len(expanded_df):,} точек")
# Проверяем финальное распределение
final_class_counts =
expanded_df['class'].value_counts().sort_index()
    print("\nРаспределение по классам (финальное):")
    for cls, count in final_class_counts.items():
        print(f" Класс {cls}: {count:,} точек")
    print()
    # Сохраняем расширенную выборку
    print(f"Сохранение расширенной выборки в
{output_csv}...")
    expanded_df.to_csv(output_csv, index=False)
    print(f" [OK] Сохранено {len(expanded_df):,} точек")
    print()
    print("=" * 70)
    print("РАСШИРЕНИЕ ЗАВЕРШЕНО")
    print("=" * 70)
    print(f"\nИсходная выборка: {len(df):,} точек")
    print(f"Расширенная выборка: {len(expanded_df):,} точек")
    print(f"Увеличение: {len(expanded_df) / len(df):.1f}x")
    print()
    if __name__ == "__main__":
        input_csv = Path(config.get_training_samples_path())
        output_csv = input_csv.parent /
"training_samples_expanded.csv"
        # Увеличиваем выборку в 5 раз
        expand_training_samples(
            input_csv=input_csv,
            output_csv=output_csv,
            expansion_factor=5, # Увеличиваем в 5 раз (500 → 2500 на
класс)
            max_neighbors=3 # Создаем точки в радиусе 3 пикселей
        )
        print(f"\nРасширенная выборка сохранена: {output_csv}")
        print("\nСледующий шаг: обновите config.py, указав:")
        print(f" TRAINING_SAMPLES_FILE =
'training_samples_expanded.csv'")

```

После того как вы вставили скрипт в файл .txt, сохраните изменения и измените формат на Python File - (.py) (рисунок 3.3)

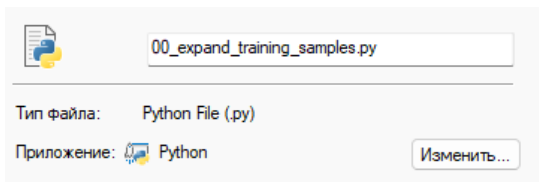


Рисунок 3.3 - Пример файла формата (.py)

Шаг - 3.4. Запуск расширения выборки

1: Откройте PowerShell и перейдите в папку MLP:

`cd <ПУТЬ_К_ПАПКЕ_MLP>`

2: Запустите скрипт расширения:

`python scripts\00_expand_training_samples.py`

Что делает скрипт:

- Читает исходную выборку `training\_samples.csv`
- Создает новые точки вокруг существующих
- Увеличивает выборку в 5 раз (500 → 2500 точек на класс)
- Сохраняет результат в `training\_samples\_expanded.csv`
- Фильтрует точки по границам области (если указан

shapefile)

Результат:

Исходная выборка: 2,500 точек

Расширенная выборка: 12,500 точек

Увеличение: 5.0x

Шаг - 3.5. Настройка параметров расширения

Если нужно изменить параметры расширения, откройте `scripts/00\_expand\_training\_samples.py` и измените:

```
python
expand_training_samples(
    input_csv=input_csv,
    output_csv=output_csv,
    expansion_factor=5, # Во сколько раз увеличить (5 = в 5 раз)
    max_neighbors=3    # Радиус в пикселях для создания
    новых точек
)
```

Рекомендации:

- `expansion\_factor=5` - стандартное значение (500 → 2500 на класс)
- `expansion\_factor=10` - для максимальной точности (500 → 5000 на класс)
- `max\_neighbors=3` - стандартное значение
- `max\_neighbors=5` - для большего разнообразия точек

Шаг - 4. НАСТРОЙКА ПАРАМЕТРОВ МОДЕЛИ

Шаг - 4.1. Файл конфигурации

Создайте файл `config` в папке проекта в формате .txt.

Откройте его в текстовом редакторе. И вставьте скрипт

ниже:

```
"""Конфигурация для проекта классификации LULC с
использованием CNN"""
import os
import sys
sys.stdout.reconfigure(encoding='utf-8')
# Базовый путь к проекту
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
# Пути к данным
DATA_DIR = os.path.join(BASE_DIR, 'data')
SATELLITE_DIR = os.path.join(DATA_DIR, 'satellite')
TRAINING_SAMPLES_DIR = os.path.join(DATA_DIR,
'training_samples')
DEM_DIR = os.path.join(DATA_DIR, 'dem')
NIGHTLIGHTS_DIR = os.path.join(DATA_DIR, 'nightlights')
# Пути к результатам
RESULTS_DIR = os.path.join(BASE_DIR, 'results')
MODELS_DIR = os.path.join(RESULTS_DIR, 'models')
PREDICTIONS_DIR = os.path.join(RESULTS_DIR,
'predictions')
REPORTS_DIR = os.path.join(RESULTS_DIR, 'reports')
# Имена файлов (пользователь должен указать в этих
переменных)
SATELLITE_FILE =
'Bulandy_Composite_2000_Full6Bands.tif' # Имя файла
спутниковых данных (GeoTIFF)
```

```

TRAINING_SAMPLES_FILE =
'training_samples_expanded.csv' # Имя файла выборки
(расширенная версия)
DEM_FILE = 'Bulandy_SRTM.tif' # Имя файла DEM
(опционально)
NIGHTLIGHTS_FILE =
'DMSP_OLS_2000_Bulandy_processed.tif' # Имя файла ночных
данных DMSP-OLS (опционально)
BOUNDARIES_FILE = os.path.join(DATA_DIR, 'boundaries',
'bulandy_boundaries_from_raster.shp') # Путь к shapefile с
границами Буланды
# Функции для получения путей
def get_satellite_path():
    """Возвращает путь к файлу спутниковых данных"""
    return os.path.join(SATELLITE_DIR, SATELLITE_FILE)
def get_training_samples_path():
    """Возвращает путь к файлу обучающей выборки"""
    return os.path.join(TRAINING_SAMPLES_DIR,
TRAINING_SAMPLES_FILE)
def get_dem_path():
    """Возвращает путь к DEM файлу (если указан)"""
    if DEM_FILE is None:
        return None
    return os.path.join(DEM_DIR, DEM_FILE)
def get_nightlights_path():
    """Возвращает путь к файлу ночных данных DMSP-OLS
(если указан)"""
    if NIGHTLIGHTS_FILE is None:
        return None
    return os.path.join(NIGHTLIGHTS_DIR,
NIGHTLIGHTS_FILE)
def get_boundaries_path():
    """Возвращает путь к shapefile с границами Буланды
(если указан)"""
    if BOUNDARIES_FILE is None:
        return None
    return BOUNDARIES_FILE
# Параметры модели CNN (Convolutional Neural Network)

```

```

# Используем улучшенную модель с attention механизмами
USE_IMPROVED_CNN = True # Использовать
улучшенную CNN с attention
CNN_CONFIG = {
    'input_channels': None, # Будет определено автоматически
    'num_classes': 6,
    'patch_size': 7, # Оптимальный размер патча для точного
разделения классов (7x7 пикселей)
    'hidden_channels': [128, 256, 512, 512], # Глубокая
архитектура с attention (4 блока)
    'dropout': 0.3, # Оптимальный dropout
    'use_batch_norm': True,
    'activation': 'leaky_relu', # LeakyReLU лучше для
различения похожих классов
    'use_attention': True # Использовать attention механизмы
}
# Параметры для ensemble (если используется)
USE_ENSEMBLE = True
ENSEMBLE_SIZE = 5 # Количество моделей в ensemble
NUM_ENSEMBLE_MODELS = ENSEMBLE_SIZE
# Параметры обучения
LEARNING_RATE = 0.0008 # Немного уменьшен для более
стабильного обучения
BATCH_SIZE = 48 # Оптимальный batch size
NUM_EPOCHS = 400 # Больше эпох для лучшего обучения
WEIGHT_DECAY = 1.2e-4 # Немного увеличен для
регуляризации
EARLY_STOPPING_PATIENCE = 40 # Больше терпения
для early stopping
GRADIENT_CLIP = 1.0
USE_CLASS_WEIGHTS = True
USE_FOCAL_LOSS = True
USE_OVERSAMPLING = True
USE_WEIGHTED_SAMPLER = True # Использовать
WeightedRandomSampler для балансировки батчей
# Параметры данных
AUGMENT_DATA = True

```

```

USE_CONTEXT_FEATURES = True # Для CNN это
означает использование патчей
CONTEXT_WINDOW_SIZE = 7 # Размер патча (должен
совпадать с patch_size)
# Параметры для работы с данными
USE_INDICES = True # Использовать вегетационные
индексы
USE_DEM = True # Использовать DEM
USE_NIGHTLIGHTS = True # Использовать ночные данные
# Параметры для сохранения моделей
BEST_MODEL_NAME = 'best_cnn_model.pth'
FINAL_MODEL_NAME = 'cnn_final_model.pth'
# Классы LULC (названия для отчетов)
LULC_CLASSES = {
    0: 'Фон',
    1: 'Пашня',
    2: 'Пастбища',
    3: 'Леса',
    4: 'Вода',
    5: 'Застроенные территории'
}
# Устройство для вычислений (CPU или GPU)
import torch
DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
# Параметры разделения данных (train/val/test)
TRAIN_RATIO = 0.7 # 70% для обучения
VAL_RATIO = 0.15 # 15% для валидации
TEST_RATIO = 0.15 # 15% для тестирования

```

После того как вы вставили скрипт в файл .txt, сохраните изменения и измените формат на Python File (.py). (рисунок 3.4)

	__pycache__	09.02.2026 02:51	Папка с файлами
	data	27.01.2026 12:25	Папка с файлами
	models	27.01.2026 12:25	Папка с файлами
	results	27.01.2026 12:25	Папка с файлами
	scripts	12.02.2026 10:14	Папка с файлами
	config.py	12.02.2026 10:44	Python File 8 КБ

Рисунок 3.4 - Созданный файл config.py

Шаг - 4.2. Параметры обучения

#####LEARNING_RATE (скорость обучения)

Расположение: `config.py`, строка ~12

```
```python
LEARNING_RATE = 0.0005
```
```

Что делает: Определяет размер шага при обновлении весов модели.

Как изменить:

1. Откройте `config.py`
2. Найдите строку `LEARNING\_RATE = 0.0005`
3. Измените значение

Рекомендации:

- Низкая точность (< 60%): Увеличьте до `0.001` или `0.002`
- Модель не обучается: Увеличьте до `0.001`
- Переобучение: Уменьшите до `0.0003`
- Стандартное значение: `0.0005`

Типичные значения: 0.0001 - 0.002

BATCH_SIZE (размер батча)

Расположение: `config.py`, строка ~13

```
```python
BATCH_SIZE = 64
```
```

Что делает: Количество образцов, обрабатываемых за одну итерацию.

Как изменить:

1. Откройте `config.py`
2. Найдите строку `BATCH\_SIZE = 64`
3. Измените значение

Рекомендации:

- Медленное обучение: Увеличьте до `128` или `256` (если есть GPU)

- Недостаточно памяти: Уменьшите до `32` или `16`
- Стандартное значение: `64`

Типичные значения: 32 - 256

NUM_EPOCHS (количество эпох)

Расположение: `config.py`, строка ~14

```
```python
```

```
NUM_EPOCHS = 200
'''
```

Что делает: Максимальное количество проходов по всей обучающей выборке.

Как изменить:

1. Откройте `config.py`
2. Найдите строку `NUM\_EPOCHS = 200`
3. Измените значение

Рекомендации:

- Ранняя остановка срабатывает слишком рано: Увеличьте до `300` или `400`

- Модель переобучается: Early stopping автоматически остановит обучение

- Стандартное значение: `200`

Типичные значения: 100 - 400

#### WEIGHT\_DECAY (L2 регуляризация)

Расположение: `config.py`, строка ~15

```
'''python
```

```
WEIGHT_DECAY = 1e-4
```

```
'''
```

Что делает: Предотвращает переобучение, штрафуя большие веса.

Как изменить:

1. Откройте `config.py`
2. Найдите строку `WEIGHT\_DECAY = 1e-4`
3. Измените значение

Рекомендации:

- Переобучение: Увеличьте до `2e-4` или `5e-4`

- Недообучение: Уменьшите до `5e-5`

- Стандартное значение: `1e-4`

Типичные значения: 1e-5 - 5e-4

#### EARLY\_STOPPING\_PATIENCE

Расположение: `config.py`, строка ~16

```
'''python
```

```
EARLY_STOPPING_PATIENCE = 30
```

```
'''
```

Что делает: Количество эпох без улучшения перед остановкой обучения.

Как изменить:

1. Откройте `config.py`
2. Найдите строку `EARLY\_STOPPING\_PATIENCE = 30`
3. Измените значение

Рекомендации:

- Обучение останавливается слишком рано: Увеличьте до `40` или `50`

- Модель переобучается: Уменьшите до `20`

- Стандартное значение: `30`

Типичные значения: 20 - 50

#### Шаг - 4.3. Параметры архитектуры модели

##### hidden\_channels (размеры скрытых слоев)

**\*\*Расположение:\*\*** `config.py`, строка ~30

```python

```
CNN_CONFIG = {  
    'hidden_channels': [512, 256, 128, 64],
```

```
    ...
```

```
}
```

```
```
```

**\*\*Что делает:\*\*** Определяет архитектуру сети (количество нейронов в каждом слое).

**\*\*Как изменить:\*\***

1. Откройте `config.py`
2. Найдите словарь `CNN\_CONFIG`
3. Измените список `hidden\_channels`

**\*\*Рекомендации:\*\***

- **\*\*Низкая точность (< 60%):\*\*** Увеличьте: `[1024, 512, 256, 128, 64]`

- **\*\*Переобучение:\*\*** Уменьшите: `[256, 128, 64]`

- **\*\*Стандартное значение:\*\*** `[512, 256, 128, 64]`

**\*\*Примеры архитектур:\*\***

- Малая: `[256, 128, 64]` - быстрее, меньше памяти

- Средняя: `[512, 256, 128, 64]` - баланс скорости и точности

- Большая: `[1024, 512, 256, 128, 64]` - выше точность,

больше памяти

---

##### dropout (вероятность dropout)

**\*\*Расположение:\*\*** `config.py`, строка ~32

```

```python
CNN_CONFIG = {
    'dropout': 0.2,
    ...
}
```

```

**Что делает:** Вероятность отключения нейрона для предотвращения переобучения.

**Как изменить:**

1. Откройте `config.py`
2. Найдите словарь `CNN\_CONFIG`
3. Измените значение `dropout`

**Рекомендации:**

- **Переобучение:** Увеличьте до `0.3` или `0.4`
- **Недообучение:** Уменьшите до `0.1`
- **Стандартное значение:** `0.2`
- **Типичные значения:** 0.1 - 0.5

---

```

use_batch_norm (batch normalization)
Расположение: `config.py`, строка ~33
```python
CNN_CONFIG = {
    'use_batch_norm': True,
    ...
}
```

```

**Что делает:** Нормализует входы каждого слоя для ускорения обучения.

**Рекомендации:**

- **Всегда рекомендуется:** Оставьте `True`
- **Проблемы с памятью:** Можно установить `False`, но точность может снизиться

---

```

use_residual (residual connections)
Расположение: `config.py`, строка ~34
```python
CNN_CONFIG = {
    'use_residual': True,

```

```
...
}
...

```

****Что делает:**** Добавляет остаточные связи для облегчения обучения глубоких сетей.

****Рекомендации:****

- ****Всегда рекомендуется:**** Оставьте `True`

- ****Проблемы с памятью:**** Можно установить `False`

```
---
```

Шаг - 4.4. Параметры для работы с данными

USE_CLASS_WEIGHTS (веса классов)

Расположение: `config.py`, строка ~18

```
```python
```

```
USE_CLASS_WEIGHTS = True
```

```
```
```

Что делает: Увеличивает важность редких классов в функции потерь.

Рекомендации:

- Несбалансированные классы: Всегда `True`

- Сбалансированные классы: Можно `False`, но обычно лучше `True`

USE_FOCAL_LOSS (Focal Loss)

Расположение: `config.py`, строка ~19

```
```python
```

```
USE_FOCAL_LOSS = False
```

```
```
```

Что делает: Фокусируется на сложных примерах для обучения.

Рекомендации:

- Сложные классы плохо классифицируются: Установите `True`

- Хорошая точность по всем классам: Можно `False`

Примечание: Для CNN рекомендуется `False` (используется обычный CrossEntropyLoss)

USE_OVERSAMPLING (oversampling)

Расположение: `config.py`, строка ~20

```
```python
```

```
USE_OVERSAMPLING = True
```

```
'''
```

Что делает: Увеличивает количество образцов редких классов.

Рекомендации:

- Редкие классы (< 10% выборки): Оставьте `True`
- Все классы хорошо представлены: Можно `False`

#### Шаг - 4.5. Параметры признаков

```
patch_size (размер окна для контекста)
```

```
Расположение: `config.py`, строка ~114
```

```
```python
```

```
patch_size = 3
```

```
'''
```

Что делает: Размер окна для извлечения контекстных признаков (3x3, 5x5, и т.д.).

Как изменить:

1. Откройте `config.py`
2. Найдите строку `patch\_size = 3`
3. Измените значение

Рекомендации:

- **Мелкие объекты плохо классифицируются:** Увеличьте до `5` или `7`

- **Медленная обработка:** Уменьшите до `3` (минимум)

- **Стандартное значение:** `3`

Типичные значения: 3, 5, 7 (только нечетные числа)

```
---
```

```
##### BOUNDARIES_FILE (путь к shapefile)
```

```
**Расположение:** `config.py`, строка ~69
```

```
```python
```

```
BOUNDARIES_FILE = os.path.join(DATA_DIR, 'boundaries',
'bulandy_boundaries_from_raster.shp')
```

```
'''
```

**Что делает:** Указывает путь к shapefile с границами области исследования.

**Как изменить:**

1. Откройте `config.py`
2. Найдите строку `BOUNDARIES\_FILE = ...`
3. Укажите путь к вашему shapefile

**Пример:**

```

```python
BOUNDARIES_FILE = os.path.join(DATA_DIR, 'boundaries',
'your_boundaries.shp')
```

```

**\*\*Преимущества:\*\***

- Сокращение времени обучения
- Повышение точности (нет шума от внешних областей)
- Корректная классификация только нужной области

---

#### **Шаг - 4.6. Создание моделей глубокого обучения**

Этап 1: Задача конфигурации для дальнейшей работы по модели.

Создайте файл формата .txt в папке models и назовите его `__init__`. Далее откройте файл через «Блокнот» и вставьте скрипт:

```

"""Модели для классификации LULC"""
from .cnn import CNN
__all__ = ['CNN']

```

После того как вы вставили скрипт в файл .txt, сохраните изменения и измените формат на Python File (.py)

Этап 2: Создание модели CNN.

Создайте файл формата .txt в папке models и назовите его `cnn`. Далее откройте файл через «Блокнот» и вставьте скрипт:

```

"""Архитектура CNN для классификации LULC"""
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.nn.init as init
class CNN(nn.Module):
 """

```

Convolutional Neural Network для классификации LULC

Использует патчи (окна) вокруг пикселя для извлечения пространственных признаков.

```

 """
 def __init__(self, input_channels, num_classes, patch_size=7,
hidden_channels=[64, 128, 256],
dropout=0.3, use_batch_norm=True, activation='relu'):
 """

```

Args:

input\_channels: количество входных каналов (спутниковые + индексы + DEM + ночные данные)

num\_classes: количество классов для классификации

patch\_size: размер патча (patch\_size x patch\_size)

hidden\_channels: список количества каналов в сверточных слоях

dropout: вероятность dropout

use\_batch\_norm: использовать ли batch normalization

activation: функция активации ('relu', 'leaky\_relu', 'gelu')

"""

super(CNN, self).\_\_init\_\_()

self.patch\_size = patch\_size

self.input\_channels = input\_channels

# Определяем функцию активации

if activation == 'relu':

self.activation = nn.ReLU()

elif activation == 'leaky\_relu':

self.activation = nn.LeakyReLU(0.2)

elif activation == 'gelu':

self.activation = nn.GELU()

else:

raise ValueError(f"Unknown activation function:

{activation}")

# Сверточные слои с depthwise separable convolutions

для лучшей генерализации

layers = []

in\_channels = input\_channels

for i, out\_channels in enumerate(hidden\_channels):

# Используем два последовательных сверточных слоя

для увеличения глубины

# Первый сверточный слой

layers.append(nn.Conv2d(in\_channels, out\_channels,

kernel\_size=3, padding=1))

if use\_batch\_norm:

layers.append(nn.BatchNorm2d(out\_channels))

layers.append(self.activation)

# Второй сверточный слой для увеличения глубины

```

 layers.append(nn.Conv2d(out_channels, out_channels,
kernel_size=3, padding=1))
 if use_batch_norm:
 layers.append(nn.BatchNorm2d(out_channels))
 layers.append(self.activation)
 # Третий сверточный слой с kernel_size=1 для
увеличения нелинейности
 if i < len(hidden_channels) - 1: # Не применяем к
последнему блоку
 layers.append(nn.Conv2d(out_channels, out_channels,
kernel_size=1))
 if use_batch_norm:
 layers.append(nn.BatchNorm2d(out_channels))
 layers.append(self.activation)
 # Dropout после каждого блока
 if dropout > 0:
 layers.append(nn.Dropout2d(dropout))
 in_channels = out_channels
 self.conv_layers = nn.Sequential(*layers)
 # Вычисляем размер после сверточных слоев
 # После каждого слоя размер уменьшается на 2 (если
padding=1 и kernel=3, размер сохраняется)
 # Но мы используем padding=1, поэтому размер патча
сохраняется
 # После всех сверточных слоев применяем глобальный
average pooling
 self.global_pool = nn.AdaptiveAvgPool2d(1)
 # Полносвязные слои с attention-like механизмом для
фокусировки на важных признаках
 self.fc1 = nn.Linear(hidden_channels[-1], 512)
 self.bn1 = nn.BatchNorm1d(512) if use_batch_norm else
nn.Identity()
 self.fc2 = nn.Linear(512, 256)
 self.bn2 = nn.BatchNorm1d(256) if use_batch_norm else
nn.Identity()
 self.fc3 = nn.Linear(256, 128)
 self.bn3 = nn.BatchNorm1d(128) if use_batch_norm else
nn.Identity()

```

```

 self.fc_out = nn.Linear(128, num_classes)
 self.dropout1 = nn.Dropout(dropout * 0.7) if dropout > 0
 else nn.Identity()
 self.dropout2 = nn.Dropout(dropout * 0.5) if dropout > 0
 else nn.Identity()
 self.dropout3 = nn.Dropout(dropout * 0.3) if dropout > 0
 else nn.Identity()
 # Инициализация весов
 self._initialize_weights()
 def _initialize_weights(self):
 """Инициализация весов сети"""
 for m in self.modules():
 if isinstance(m, nn.Conv2d):
 # Для GELU используем kaiming_normal с правильным
nonlinearity
 if isinstance(self.activation, nn.GELU):
 init.kaiming_normal_(m.weight, mode='fan_out',
nonlinearity='relu')
 else:
 init.kaiming_normal_(m.weight, mode='fan_out',
nonlinearity='relu')
 if m.bias is not None:
 init.constant_(m.bias, 0)
 elif isinstance(m, nn.BatchNorm2d) or isinstance(m,
nn.BatchNorm1d):
 init.constant_(m.weight, 1)
 init.constant_(m.bias, 0)
 elif isinstance(m, nn.Linear):
 init.xavier_uniform_(m.weight)
 if m.bias is not None:
 init.constant_(m.bias, 0)
 def forward(self, x):
 """
 Forward pass
 Args:
 x: входной тензор формы (batch_size, input_channels,
patch_size, patch_size)
 Returns:

```

```

выходной тензор формы (batch_size, num_classes)
"""
Сверточные слои
x = self.conv_layers(x)
Глобальный average pooling
x = self.global_pool(x)
Flatten
x = x.view(x.size(0), -1)
Полносвязные слои с residual-like connections
x1 = self.fc1(x)
x1 = self.bn1(x1)
x1 = self.activation(x1)
x1 = self.dropout1(x1)
x2 = self.fc2(x1)
x2 = self.bn2(x2)
x2 = self.activation(x2)
x2 = self.dropout2(x2)
x3 = self.fc3(x2)
x3 = self.bn3(x3)
x3 = self.activation(x3)
x3 = self.dropout3(x3)
Выходной слой
x_out = self.fc_out(x3)
return x_out

```

После того как вы вставили скрипт в файл .txt, сохраните изменения и измените формат на Python File (.py)

Этап 3: Задача конфигурации для дальнейшей работы по модели.

Создайте файл формата .txt в папке models и назовите его cnn\_improved. Далее откройте файл через «Блокнот» и вставьте скрипт:

```

"""Улучшенная архитектура CNN с attention механизмами
для классификации LULC"""
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.nn.init as init

```

```

class ChannelAttention(nn.Module):
 """Attention механизм для каналов (Channel Attention)"""
 def __init__(self, num_channels, reduction=16):
 super(ChannelAttention, self).__init__()
 self.avg_pool = nn.AdaptiveAvgPool2d(1)
 self.max_pool = nn.AdaptiveMaxPool2d(1)
 self.fc = nn.Sequential(
 nn.Linear(num_channels, num_channels // reduction,
bias=False),
 nn.ReLU(),
 nn.Linear(num_channels // reduction, num_channels,
bias=False)
)
 self.sigmoid = nn.Sigmoid()

 def forward(self, x):
 b, c, _, _ = x.size()
 # Average pooling
 avg_out = self.fc(self.avg_pool(x).view(b, c))
 # Max pooling
 max_out = self.fc(self.max_pool(x).view(b, c))
 out = avg_out + max_out
 out = self.sigmoid(out).view(b, c, 1, 1)
 return x * out

class SpatialAttention(nn.Module):
 """Attention механизм для пространственных признаков
(Spatial Attention)"""
 def __init__(self, kernel_size=7):
 super(SpatialAttention, self).__init__()
 self.conv = nn.Conv2d(2, 1, kernel_size=kernel_size,
padding=kernel_size//2, bias=False)
 self.sigmoid = nn.Sigmoid()

 def forward(self, x):
 # Average и Max по каналам
 avg_out = torch.mean(x, dim=1, keepdim=True)
 max_out, _ = torch.max(x, dim=1, keepdim=True)

```

```

 out = torch.cat([avg_out, max_out], dim=1)
 out = self.conv(out)
 out = self.sigmoid(out)
 return x * out

class ResidualBlock(nn.Module):
 """Residual блок с attention"""
 def __init__(self, in_channels, out_channels,
use_attention=True, activation='leaky_relu'):
 super(ResidualBlock, self).__init__()

 if activation == 'relu':
 self.activation = nn.ReLU()
 elif activation == 'leaky_relu':
 self.activation = nn.LeakyReLU(0.2)
 elif activation == 'gelu':
 self.activation = nn.GELU()
 else:
 self.activation = nn.ReLU()

 self.conv1 = nn.Conv2d(in_channels, out_channels,
kernel_size=3, padding=1)
 self.bn1 = nn.BatchNorm2d(out_channels)
 self.conv2 = nn.Conv2d(out_channels, out_channels,
kernel_size=3, padding=1)
 self.bn2 = nn.BatchNorm2d(out_channels)

 # Projection layer если размер каналов меняется
 self.projection = None
 if in_channels != out_channels:
 self.projection = nn.Sequential(
 nn.Conv2d(in_channels, out_channels, kernel_size=1),
 nn.BatchNorm2d(out_channels)
)

 # Attention механизмы
 self.use_attention = use_attention
 if use_attention:

```

```

 self.channel_attention = ChannelAttention(out_channels)
 self.spatial_attention = SpatialAttention()

 def forward(self, x):
 residual = x

 out = self.conv1(x)
 out = self.bn1(out)
 out = self.activation(out)
 out = self.conv2(out)
 out = self.bn2(out)

 # Применяем projection если нужно
 if self.projection is not None:
 residual = self.projection(residual)

 # Residual connection
 out = out + residual

 # Attention
 if self.use_attention:
 out = self.channel_attention(out)
 out = self.spatial_attention(out)

 out = self.activation(out)
 return out

class ImprovedCNN(nn.Module):
 """
 Улучшенная CNN с residual connections и attention
 механизмами

```

Ключевые улучшения:

- Residual connections для лучшего обучения глубоких сетей
- Channel Attention для фокусировки на важных каналах
- Spatial Attention для фокусировки на важных пространственных областях

- Более глубокая архитектура  
"""

```
def __init__(self, input_channels, num_classes, patch_size=7,
 hidden_channels=[128, 256, 512, 512], dropout=0.3,
 use_batch_norm=True, activation='leaky_relu',
 use_attention=True):
 """
```

Args:

input\_channels: количество входных каналов  
num\_classes: количество классов  
patch\_size: размер патча  
hidden\_channels: список каналов в сверточных слоях  
dropout: вероятность dropout  
use\_batch\_norm: использовать batch normalization  
activation: функция активации  
use\_attention: использовать attention механизмы  
"""

```
super(ImprovedCNN, self).__init__()
```

```
self.patch_size = patch_size
self.input_channels = input_channels
self.use_attention = use_attention
```

```
Определяем функцию активации
```

```
if activation == 'relu':
 self.activation = nn.ReLU()
elif activation == 'leaky_relu':
 self.activation = nn.LeakyReLU(0.2)
elif activation == 'gelu':
 self.activation = nn.GELU()
else:
 self.activation = nn.ReLU()
```

```
Первый сверточный слой (увеличение каналов)
```

```
self.initial_conv = nn.Sequential(
 nn.Conv2d(input_channels, hidden_channels[0],
 kernel_size=3, padding=1),
```

```

 nn.BatchNorm2d(hidden_channels[0]) if
use_batch_norm else nn.Identity(),
 self.activation
)

 # Residual блоки с attention
 self.residual_blocks = nn.ModuleList()
 in_channels = hidden_channels[0]

 for i, out_channels in enumerate(hidden_channels):
 self.residual_blocks.append(
 ResidualBlock(
 in_channels,
 out_channels,
 use_attention=use_attention and (i > 0), # Attention начиная со
второго блока
 activation=activation
)
)
 in_channels = out_channels

 # Глобальный average pooling
 self.global_pool = nn.AdaptiveAvgPool2d(1)

 # Полносвязные слои с dropout
 self.fc_layers = nn.Sequential(
 nn.Linear(hidden_channels[-1], 1024),
 nn.BatchNorm1d(1024) if use_batch_norm else
nn.Identity(),
 self.activation,
 nn.Dropout(dropout * 0.8),
 nn.Linear(1024, 512),
 nn.BatchNorm1d(512) if use_batch_norm else
nn.Identity(),
 self.activation,
 nn.Dropout(dropout * 0.6),
 nn.Linear(512, 256),

```

```

 nn.BatchNorm1d(256) if use_batch_norm else
nn.Identity(),
 self.activation,
 nn.Dropout(dropout * 0.4),
 nn.Linear(256, num_classes)
)

 # Инициализация весов
 self._initialize_weights()

 def _initialize_weights(self):
 """Инициализация весов"""
 for m in self.modules():
 if isinstance(m, nn.Conv2d):
 if isinstance(self.activation, (nn.LeakyReLU,
nn.ReLU)):
 init.kaiming_normal_(m.weight, mode='fan_out',
nonlinearity='relu')
 else:
 init.kaiming_normal_(m.weight, mode='fan_out',
nonlinearity='relu')
 if m.bias is not None:
 init.constant_(m.bias, 0)
 elif isinstance(m, (nn.BatchNorm2d, nn.BatchNorm1d)):
 init.constant_(m.weight, 1)
 init.constant_(m.bias, 0)
 elif isinstance(m, nn.Linear):
 init.xavier_uniform_(m.weight)
 if m.bias is not None:
 init.constant_(m.bias, 0)

 def forward(self, x):
 """
 Forward pass

 Args:
 x: входной тензор (batch_size, input_channels,
patch_size, patch_size)

```

```

Returns:
 выходной тензор (batch_size, num_classes)
"""
Начальный сверточный слой
x = self.initial_conv(x)

Residual блоки
for block in self.residual_blocks:
 x = block(x)

Глобальный average pooling
x = self.global_pool(x)
x = x.view(x.size(0), -1)
Полносвязные слои
x = self.fc_layers(x)
return x

```

После того как вы вставили скрипт в файл .txt, сохраните изменения и измените формат на Python File (.py)

## Шаг - 5. ОБУЧЕНИЕ ОДИНОЧНОЙ МОДЕЛИ

### Шаг - 5.1. Подготовка к обучению

Создайте файл формата .txt в папке scripts и назовите его 01\_check\_training\_samples. Далее откройте файл через «Блокнот» и вставьте скрипт:

```

"""Проверка формата ручной выборки перед обучением"""
import sys
sys.stdout.reconfigure(encoding='utf-8')

import os
import pandas as pd
import rasterio
from pathlib import Path

Добавляем родительскую директорию в путь для импорта
config

```

```

sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import config

def check_csv_format(csv_path):
 """Проверка формата CSV файла"""
 print("=" * 70)
 print("ПРОВЕРКА ФОРМАТА CSV ФАЙЛА")
 print("=" * 70)
 print()

 csv_path = Path(csv_path)

 if not csv_path.exists():
 print(f"[ОШИБКА] Файл не найден: {csv_path}")
 return False

 print(f"Файл: {csv_path}")
 print()

 # Проверка 1: Чтение файла
 try:
 df = pd.read_csv(csv_path, encoding='utf-8')
 print("[ОК] Файл успешно прочитан")
 except UnicodeDecodeError:
 try:
 df = pd.read_csv(csv_path, encoding='cp1251')
 print("[WARNING] Файл прочитан с кодировкой
cp1251, рекомендуется UTF-8")
 except:
 print("[ОШИБКА] Не удалось прочитать файл.
Проверьте кодировку (должна быть UTF-8)")
 return False

 except Exception as e:
 print(f"[ОШИБКА] Ошибка при чтении файла: {e}")
 return False

```

# Проверка 2: Наличие необходимых колонок

```

print("\nПроверка структуры файла...")
required_columns = ['class', 'x', 'y']
missing_columns = [col for col in required_columns if col not
in df.columns]

```

```

if missing_columns:
 print(f"[ОШИБКА] Отсутствуют необходимые колонки:
{missing_columns}")
 print(f" Найдены колонки: {list(df.columns)}")
 return False

```

```

print(f"[OK] Найдены все необходимые колонки:
{required_columns}")

```

```

Проверка 3: Типы данных
print("\nПроверка типов данных...")

```

```

if not pd.api.types.is_numeric_dtype(df['class']):
 print("[ОШИБКА] Колонка 'class' должна содержать
числа")
 return False
print("[OK] Колонка 'class' содержит числа")

```

```

if not pd.api.types.is_numeric_dtype(df['x']):
 print("[ОШИБКА] Колонка 'x' должна содержать
числа")
 return False
print("[OK] Колонка 'x' содержит числа")

```

```

if not pd.api.types.is_numeric_dtype(df['y']):
 print("[ОШИБКА] Колонка 'y' должна содержать
числа")
 return False
print("[OK] Колонка 'y' содержит числа")

```

```

Проверка 4: Пропущенные значения
print("\nПроверка пропущенных значений...")
missing_count = df[required_columns].isnull().sum().sum()

```

```

 if missing_count > 0:
 print(f"[WARNING] Найдено {missing_count}
пропущенных значений")
 print(" Эти строки будут удалены при загрузке")
 df = df.dropna(subset=required_columns)
 else:
 print("[OK] Пропущенных значений нет")

Проверка 5: Диапазон значений классов
print("\nПроверка классов...")
unique_classes = sorted(df['class'].unique())
print(f"Найдены классы: {unique_classes}")

Проверяем, что классы в разумном диапазоне (1-5 или
0-5)
invalid_classes = [c for c in unique_classes if c < 0 or c > 5]
if invalid_classes:
 print(f"[WARNING] Найдены классы вне диапазона 0-5:
{invalid_classes}")
 print(" Ожидаются классы: 0 (фон), 1-5 (классы
LULC)")
else:
 print("[OK] Все классы в допустимом диапазоне")

Проверка 6: Диапазон координат
print("\nПроверка координат...")
x_min, x_max = df['x'].min(), df['x'].max()
y_min, y_max = df['y'].min(), df['y'].max()

print(f"Долгота (x): {x_min:.6f} - {x_max:.6f}")
print(f"Широта (y): {y_min:.6f} - {y_max:.6f}")

Проверяем разумность координат (для Казахстана
примерно 46-55°N, 46-88°E)
if x_min < -180 or x_max > 180:
 print("[WARNING] Долгота вне диапазона -180 до 180
градусов")
if y_min < -90 or y_max > 90:

```

```

 print("[WARNING] Широта вне диапазона -90 до 90
градусов")

Проверка 7: Статистика по классам
print("\nСтатистика по классам:")
print("-" * 70)
class_counts = df['class'].value_counts().sort_index()
for class_id, count in class_counts.items():
 class_name = config.LULC_CLASSES.get(int(class_id),
fКласс {int(class_id)})
 percentage = count / len(df) * 100
 print(f" Класс {int(class_id)} ({class_name}): {count}
точек ({percentage:.1f}%)")
 print(f"\nВсего точек: {len(df)}")
 print()

Проверка 8: Соответствие координат спутниковым
данным (если указаны)
print("Проверка соответствия координат спутниковым
данным...")
try:
 satellite_path = config.get_satellite_path()
 if Path(satellite_path).exists():
 with rasterio.open(satellite_path) as src:
 bounds = src.bounds
 print(f"Границы спутниковых данных:")
 print(f" Долгота: {bounds.left:.6f} -
{bounds.right:.6f}")
 print(f" Широта: {bounds.bottom:.6f} -
{bounds.top:.6f}")

 # Проверяем, попадают ли точки в границы
 points_in_bounds = (
(df['x'] >= bounds.left) & (df['x'] <= bounds.right) &
(df['y'] >= bounds.bottom) & (df['y'] <= bounds.top)
)
 points_outside = (~points_in_bounds).sum()

```

```

 if points_outside > 0:
 print(f"[WARNING] {points_outside} точек находятся вне
границ спутниковых данных")
 print(" Эти точки будут пропущены при обучении")
 else:
 print("[OK] Все точки находятся в пределах спутниковых
данных")
 else:
 print("[INFO] Спутниковые данные не найдены,
проверка координат пропущена")
 except Exception as e:
 print(f"[INFO] Не удалось проверить соответствие
координат: {e}")

```

```

 print()
 print("=" * 70)
 print("[OK] ПРОВЕРКА ЗАВЕРШЕНА")
 print("=" * 70)
 print()
 print("Если все проверки пройдены успешно, можно
запускать обучение модели.")
 print("Если есть предупреждения, рекомендуется их
исправить перед обучением.")
 print()

```

```

 return True

```

```

def main():
 """Основная функция"""
 samples_path = config.get_training_samples_path()

 print("ПРОВЕРКА ФОРМАТА ОБУЧАЮЩЕЙ
ВЫБОРКИ")
 print()
 print(f"Путь к файлу выборки: {samples_path}")
 print()

```

```

 if not Path(samples_path).exists():

```

```

 print(f"[ОШИБКА] Файл не найден: {samples_path}")
 print()
 print("Убедитесь, что:")
 print(" 1. Файл training_samples.csv находится в папке
data/training_samples/")
 print(" 2. Имя файла указано правильно в config.py")
 return

 check_csv_format(samples_path)

if __name__ == "__main__":
 main()

```

После того как вы вставили скрипт в файл .txt, сохраните изменения и измените формат на Python File (.py).

1: Проверьте наличие всех файлов:

```

cd <ПУТЬ_К_ПАПКЕ_CNN>
python scripts\01_check_training_samples.py

```

Что проверяет скрипт:

- Наличие спутниковых данных
- Наличие обучающей выборки
- Наличие DEM (если указан)
- Наличие ночных данных (если указан)
- Наличие границ (если указан shapefile)
- Формат и корректность данных

Если все проверки пройдены: Можно переходить к обучению.

### Шаг - 5.2. Запуск обучения

Создайте файл формата .txt в папке scripts и назовите его 02\_train\_cnn\_model. Далее откройте файл через «Блокнот» и вставьте скрипт:

```

"""Обучение CNN модели для классификации LULC"""
import sys

```

```

sys.stdout.reconfigure(encoding='utf-8')

import os
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader,
WeightedRandomSampler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix
from tqdm import tqdm
import numpy as np
from pathlib import Path

Добавляем родительскую директорию в путь для импорта
parent_dir =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
sys.path.insert(0, parent_dir)
import config
Используем улучшенную модель если указано в config
if config.USE_IMPROVED_CNN:
 from models.cnn_improved import ImprovedCNN as CNN
else:
 from models.cnn import CNN
Импорт utils из той же папки
from utils import LULCDataset, load_training_samples

class FocalLoss(nn.Module):
 """Focal Loss для лучшей работы с трудными примерами
 с поддержкой label smoothing"""
 def __init__(self, alpha=None, gamma=2.5, reduction='mean',
label_smoothing=0.1):
 super(FocalLoss, self).__init__()
 self.alpha = alpha
 self.gamma = gamma
 self.reduction = reduction

```

```

self.label_smoothing = label_smoothing

def forward(self, inputs, targets):
 num_classes = inputs.size(1)
 log_probs = F.log_softmax(inputs, dim=1)

 # Label smoothing
 with torch.no_grad():
 true_dist = torch.zeros_like(log_probs)
 true_dist.fill_(self.label_smoothing / (num_classes - 1))
 true_dist.scatter_(1, targets.unsqueeze(1), 1.0 -
self.label_smoothing)
 if self.alpha is not None:
 true_dist = true_dist * self.alpha.unsqueeze(0)

 # Вычисляем cross entropy с label smoothing
 ce_loss = -(true_dist * log_probs).sum(dim=1)

 # Focal loss
 pt = torch.exp(-ce_loss)
 focal_loss = ((1 - pt) ** self.gamma) * ce_loss

 if self.reduction == 'mean':
 return focal_loss.mean()
 elif self.reduction == 'sum':
 return focal_loss.sum()
 else:
 return focal_loss

def train_cnn_model(model, train_loader, val_loader,
num_epochs, learning_rate, device, models_dir,
class_weights=None, use_focal_loss=True):
 """Обучение CNN модели"""
 model = model.to(device)

 # Используем Focal Loss или обычный CrossEntropyLoss
 if use_focal_loss:
 if class_weights is not None:

```

```

 class_weights = class_weights.to(device)
 criterion = FocalLoss(alpha=class_weights, gamma=2.5,
label_smoothing=0.1) # Label smoothing для регуляризации
 print("Используется Focal Loss с весами классов
(gamma=2.5, label_smoothing=0.1)")
 else:
 criterion = FocalLoss(gamma=2.5, label_smoothing=0.1)
 print("Используется Focal Loss (gamma=2.5,
label_smoothing=0.1)")
 else:
 # Используем веса классов если указаны
 if class_weights is not None:
 class_weights = class_weights.to(device)
 criterion = nn.CrossEntropyLoss(weight=class_weights)
 print("Используются веса классов для балансировки
loss")
 else:
 criterion = nn.CrossEntropyLoss()

Используем AdamW с weight_decay для лучшей
регуляризации
optimizer = torch.optim.AdamW(
 model.parameters(),
 lr=learning_rate,
 weight_decay=config.WEIGHT_DECAY,
 betas=(0.9, 0.999),
 eps=1e-8
)
Комбинированный scheduler: сначала cosine annealing,
потом reduce on plateau
scheduler1 =
torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,
T_max=num_epochs//2, eta_min=learning_rate*0.01)
scheduler2 =
torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
mode='min', factor=0.7, patience=8, min_lr=learning_rate*0.0001)

best_val_loss = float('inf')

```

```

 best_val_acc = 0.0
 best_model_path = os.path.join(models_dir,
config.BEST_MODEL_NAME)

 train_losses = []
 val_losses = []
 train_accs = []
 val_accs = []

 # Early stopping
 patience_counter = 0
 early_stopping_patience =
config.EARLY_STOPPING_PATIENCE
 min_epochs = 20 # Минимум эпох перед early stopping

 print("Начало обучения...")
 print(f"Всего эпох: {num_epochs}")
 print(f"Батчей в эпоху (train): {len(train_loader)}")
 print(f"Батчей в эпоху (val): {len(val_loader)}")
 print(f"Минимум эпох перед early stopping:
{min_epochs}")
 print("-" * 70)
 print()

 for epoch in range(num_epochs):
 # Обучение
 model.train()
 train_loss = 0.0
 train_preds = []
 train_labels = []

 for batch in tqdm(train_loader, desc=f'Эпоха
{epoch+1}/{num_epochs} [Обучение]', leave=True):
 inputs, labels = batch
 inputs = inputs.to(device)
 labels = labels.to(device)

 optimizer.zero_grad()

```

```

outputs = model(inputs)
loss = criterion(outputs, labels)
loss.backward()

Gradient clipping для стабильности обучения
if config.GRADIENT_CLIP > 0:
 torch.nn.utils.clip_grad_norm_(model.parameters(),
config.GRADIENT_CLIP)

optimizer.step()

train_loss += loss.item()
_, preds = torch.max(outputs, 1)
train_preds.extend(preds.cpu().numpy())
train_labels.extend(labels.cpu().numpy())

train_loss /= len(train_loader)
train_acc = accuracy_score(train_labels, train_preds)

Валидация
model.eval()
val_loss = 0.0
val_preds = []
val_labels = []

with torch.no_grad():
 for batch in tqdm(val_loader, desc=f'Эпоха
{epoch+1}/{num_epochs} [Валидация]', leave=True):
 inputs, labels = batch
 inputs = inputs.to(device)
 labels = labels.to(device)

 outputs = model(inputs)
 loss = criterion(outputs, labels)

 val_loss += loss.item()
 _, preds = torch.max(outputs, 1)
 val_preds.extend(preds.cpu().numpy())

```

```

 val_labels.extend(labels.cpu().numpy())

 val_loss /= len(val_loader)
 val_acc = accuracy_score(val_labels, val_preds)

 # Обновление learning rate
 current_lr = optimizer.param_groups[0]['lr']
 if epoch < num_epochs // 3:
 scheduler1.step()
 else:
 scheduler2.step(val_loss)

 train_losses.append(train_loss)
 val_losses.append(val_loss)
 train_accs.append(train_acc)
 val_accs.append(val_acc)

 print(f'Эпоха {epoch+1}/{num_epochs}:')
 print(f' Learning Rate: {current_lr:.6f}')
 print(f' Обучение: Loss = {train_loss:.4f}, Accuracy = {train_acc:.4f} ({train_acc*100:.2f}%)')
 print(f' Валидация: Loss = {val_loss:.4f}, Accuracy = {val_acc:.4f} ({val_acc*100:.2f}%)')

 # Сохраняем лучшую модель по валидационной
 точности
 if val_acc > best_val_acc or (val_acc == best_val_acc and
 val_loss < best_val_loss):
 if val_acc > best_val_acc:
 best_val_acc = val_acc
 best_val_loss = val_loss
 torch.save(model.state_dict(), best_model_path)
 patience_counter = 0
 print(f' [Модель сохранена:
 {config.BEST_MODEL_NAME}] (лучшая точность:
 {best_val_acc*100:.2f}%)')
 else:
 patience_counter += 1

```

```

 # Early stopping только после минимального
количества эпох
 if epoch >= min_epochs and patience_counter >=
early_stopping_patience:
 print(f'\n[EARLY STOPPING] Нет улучшения
{early_stopping_patience} эпох. Остановка обучения.')
 print(f'Лучшая точность: {best_val_acc*100:.2f}%')
 print(f'Прошло эпох: {epoch+1}')
 break
 print()

 return {
 'train_losses': train_losses,
 'val_losses': val_losses,
 'train_accs': train_accs,
 'val_accs': val_accs,
 'best_val_acc': best_val_acc
 }

def evaluate_model(model, test_loader, device):
 """Оценка модели на тестовых данных"""
 model = model.to(device)
 model.eval()

 all_preds = []
 all_labels = []

 print("Оценка на тестовых данных...")
 with torch.no_grad():
 for batch in tqdm(test_loader, desc='Оценка'):
 inputs, labels = batch
 inputs = inputs.to(device)
 labels = labels.to(device)

 outputs = model(inputs)
 _, preds = torch.max(outputs, 1)
 all_preds.extend(preds.cpu().numpy())
 all_labels.extend(labels.cpu().numpy())

```

```

accuracy = accuracy_score(all_labels, all_preds)
print(f'\nТочность на тестовых данных: {accuracy:.4f}
({accuracy*100:.2f}%)')
print()

Confusion Matrix
classes = sorted(set(all_labels + all_preds))
cm = confusion_matrix(all_labels, all_preds, labels=classes)
print('Матрица ошибок (Confusion Matrix):')
print("-" * 70)
print("Предсказано ->", end="")
for c in classes:
 print(f" Класс {c}", end="")
print()
for i, true_class in enumerate(classes):
 print(f"Истинный класс {true_class} ", end="")
 for j, pred_class in enumerate(classes):
 print(f" {cm[i, j]:6d}", end="")
 print()
print()

Classification Report
class_names = [config.LULC_CLASSES.get(i, f'Класс {i}')]
for i in classes]
print('Отчет о классификации:')
print("-" * 70)
print(classification_report(all_labels, all_preds,
 labels=classes,
 target_names=class_names,
 zero_division=0))

return accuracy, all_preds, all_labels, cm

def main():
 """Основная функция"""
 print("=" * 70)

```

```

 print("ОБУЧЕНИЕ CNN МОДЕЛИ ДЛЯ
КЛАССИФИКАЦИИ LULC")
 print("=" * 70)
 print()

 device = torch.device(config.DEVICE)
 print(f"Устройство: {device}")
 if device.type == 'cuda':
 print(f" GPU: {torch.cuda.get_device_name(0)}")
 print()

 # Проверка наличия данных
 try:
 satellite_path = config.get_satellite_path()
 except ValueError as e:
 print(f"[ОШИБКА] {e}")
 print()
 print("Укажите имя файла спутниковых данных в
config.py:")
 print(" SATELLITE_FILE = 'ваш_файл.tif")
 return

 training_samples_path = config.get_training_samples_path()
 dem_path = config.get_dem_path()
 nightlights_path = config.get_nightlights_path()

 print("Проверка файлов...")
 if not Path(satellite_path).exists():
 print(f"[ОШИБКА] Спутниковые данные не найдены:
{satellite_path}")
 return
 print(f" [OK] Спутниковые данные:
{Path(satellite_path).name}")

 if not Path(training_samples_path).exists():
 print(f"[ОШИБКА] Файл выборки не найден:
{training_samples_path}")
 print()

```

```

 print("Создайте файл training_samples.csv в папке
data/training_samples/")
 print("Формат: class,x,y (см.
data/training_samples/FORMAT.txt)")
 return
 print(f" [OK] Файл выборки:
{Path(training_samples_path).name}")

 if dem_path and Path(dem_path).exists():
 print(f" [OK] DEM: {Path(dem_path).name}")
 else:
 print(f" [INFO] DEM не указан или не найден")

 if nightlights_path and Path(nightlights_path).exists():
 print(f" [OK] Ночные данные DMSP-OLS:
{Path(nightlights_path).name}")
 else:
 print(f" [INFO] Ночные данные не указаны или не
найжены")
 print()

 # Загрузка выборки
 print("Загрузка выборки...")
 samples_df = load_training_samples(training_samples_path)

 # Загрузка границ Буланды (если указаны)
 boundaries_path = config.get_boundaries_path()
 if boundaries_path:
 print(f" [OK] Используются границы Буланды:
{Path(boundaries_path).name}")
 else:
 print(f" [INFO] Границы Буланды не указаны (можно
указать BOUNDARIES_FILE в config.py)")
 print()

 # Создание датасета с нормализацией, augmentation и
контекстными признаками
 print("Создание датасета...")

```

```

dataset = LULCDataset(
 satellite_path=satellite_path,
 samples_df=samples_df,
 dem_path=dem_path,
 nightlights_path=nightlights_path, # Ночные данные
DMSP-OLS
 normalize=True, # Включаем нормализацию для
лучшей точности
 augment=config.AUGMENT_DATA, # Включаем data
augmentation
 patch_size=config.CNN_CONFIG['patch_size'], # Размер
патча для CNN
 boundaries_path=boundaries_path # Фильтрация по
границам Буланды
)

Определяем количество входных каналов
sample_patch, _ = dataset[0]
input_channels = sample_patch.shape[0]

print(f"Количество входных каналов: {input_channels}")
print(f"Размер патча:
{config.CNN_CONFIG['patch_size']}x{config.CNN_CONFIG['patch
size']}")

Подсчитываем количество каналов по категориям
num_channels = dataset.satellite_data.shape[0]
num_indices = len(dataset.indices_data) if hasattr(dataset,
'indices_data') else 0
num_dem = 1 if dataset.dem_data is not None else 0
num_nightlights = 1 if dataset.nightlights_data is not None
else 0

print(f" Спутниковые каналы: {num_channels}")
print(f" Вегетационные индексы: {num_indices}")
print(f" DEM: {num_dem}")
print(f" Ночные данные: {num_nightlights}")
print(f" Всего каналов: {input_channels}")

```

```

print()

Разделение на train/val/test
print("Разделение данных на train/val/test...")
indices = np.arange(len(dataset))
train_indices, temp_indices = train_test_split(
 indices,
 test_size=(config.VAL_RATIO + config.TEST_RATIO),
 random_state=42,
 stratify=dataset.samples_df['class'].values
)
val_indices, test_indices = train_test_split(
 temp_indices,
 test_size=(config.TEST_RATIO / (config.VAL_RATIO +
config.TEST_RATIO)),
 random_state=42,

stratify=dataset.samples_df.iloc[temp_indices]['class'].values
)

Oversampling проблемных классов (2 и 5) в обучающей
выборке
if config.USE_OVERSAMPLING:
 print("Применение oversampling для проблемных
классов...")
 train_classes =
dataset.samples_df.iloc[train_indices]['class'].values
 class_counts = np.bincount(train_classes, minlength=6)

 # Определяем проблемные классы (класс 2 и 5)
 problem_classes = [2, 5]
 target_count = int(np.max(class_counts[1:])) # Целевое
количество = максимум среди всех классов

 original_count = len(train_indices)
 oversampled_indices = list(train_indices)

 for cls in problem_classes:

```

```

cls_mask = train_classes == cls
cls_indices = train_indices[cls_mask]
current_count = len(cls_indices)
if current_count < target_count:
 # Дублируем примеры проблемного класса
 needed = target_count - current_count
 oversample_indices = np.random.choice(cls_indices,
size=needed, replace=True)
 oversampled_indices.extend(oversample_indices)
 print(f" Класс {cls}: {current_count} →
{target_count} (добавлено {needed})")

train_indices = np.array(oversampled_indices)
print(f" Итого обучающих примеров:
{len(train_indices)} (было {original_count})")
print()

Вычисляем веса классов для балансировки loss
class_weights = None
if config.USE_CLASS_WEIGHTS:
 print("Вычисление весов классов...")
 train_classes =
dataset.samples_df.iloc[train_indices]['class'].values
 unique_classes, class_counts = np.unique(train_classes,
return_counts=True)
 total_samples = len(train_classes)
 num_classes = len(unique_classes)

 # Создаем словарь для весов
 class_weights_dict = {}
 for cls, count in zip(unique_classes, class_counts):
 # Обратно пропорциональные веса (меньше класс =
 # больше вес)
 class_weights_dict[int(cls)] = total_samples /
(num_classes * count)

 # Создаем тензор весов для всех классов (0-5)
 class_weights = torch.ones(6) # 6 классов: 0-5

```

```

for cls, weight in class_weights_dict.items():
 if 0 <= cls < 6:
 class_weights[cls] = weight

Нормализуем веса
class_weights = class_weights / class_weights.mean()

Увеличиваем вес для класса 2 (Пастбища) - самого
проблемного класса
Увеличиваем его вес в 1.8 раза (немного меньше для
избежания переобучения)
if 2 in class_weights_dict:
 class_weights[2] = class_weights[2] * 1.8

Также немного увеличиваем вес для класса 1 (Пашня)
- тоже проблемный
if 1 in class_weights_dict:
 class_weights[1] = class_weights[1] * 1.15

Нормализуем веса обратно
class_weights = class_weights / class_weights.mean()

print(f" Веса классов (с усилением для проблемных):
{dict(zip(range(6), class_weights.numpy()))}")
print()

train_dataset = torch.utils.data.Subset(dataset, train_indices)
val_dataset = torch.utils.data.Subset(dataset, val_indices)
test_dataset = torch.utils.data.Subset(dataset, test_indices)

Weighted Random Sampler для балансировки классов в
обучении
train_labels =
dataset.samples_df.iloc[train_indices]['class'].values
class_counts = np.bincount(train_labels, minlength=6)

print(f"Распределение классов в обучающей выборке:")
for cls in range(1, 6):

```

```

 count = class_counts[cls] if cls < len(class_counts) else 0
 if count > 0:
 print(f" Класс {cls}: {count} примеров")
 print()

 # Используем WeightedRandomSampler только если
 классы несбалансированы
 if hasattr(config, 'USE_WEIGHTED_SAMPLER') and
 config.USE_WEIGHTED_SAMPLER and np.std(class_counts[1:])
 > 50: # Если разница больше 50
 class_weights_sampler = 1.0 / (class_counts[1:] + 1e-6) #
 Избегаем деления на 0
 class_weights_sampler = class_weights_sampler /
 class_weights_sampler.sum() * len(class_weights_sampler)

 # Создаем веса для каждого примера
 sample_weights = np.zeros(len(train_labels))
 for i, label in enumerate(train_labels):
 if 1 <= label <= 5:
 sample_weights[i] = class_weights_sampler[label - 1]

 sampler =
 WeightedRandomSampler(weights=torch.FloatTensor(sample_weights),
 num_samples=len(train_labels),
 replacement=True)
 train_loader = DataLoader(train_dataset,
 batch_size=config.BATCH_SIZE, sampler=sampler,
 num_workers=0)
 print("Используется WeightedRandomSampler для
 балансировки")
 else:
 train_loader = DataLoader(train_dataset,
 batch_size=config.BATCH_SIZE, shuffle=True, num_workers=0)
 print("Используется обычный shuffle (классы
 сбалансированы)")
 val_loader = DataLoader(val_dataset,
 batch_size=config.BATCH_SIZE, shuffle=False, num_workers=0)

```

```

 test_loader = DataLoader(test_dataset,
batch_size=config.BATCH_SIZE, shuffle=False, num_workers=0)

 print(f"Обучающих образцов: {len(train_dataset):,}
({len(train_dataset)/len(dataset)*100:.1f}%)")
 print(f"Валидационных образцов: {len(val_dataset):,}
({len(val_dataset)/len(dataset)*100:.1f}%)")
 print(f"Тестовых образцов: {len(test_dataset):,}
({len(test_dataset)/len(dataset)*100:.1f}%)")
 print(f"Батчей в эпоху (train): {len(train_loader)},")
 print(f"Батчей в эпоху (val): {len(val_loader)},")
 print()

 # Создание модели
 print("Создание модели CNN...")
 if config.USE_IMPROVED_CNN:
 print(" [INFO] Используется улучшенная CNN с
attention механизмами")
 cnn_config = config.CNN_CONFIG.copy()
 cnn_config['input_channels'] = input_channels
 print(f"Конфигурация:")
 print(f" Входные каналы: {cnn_config['input_channels']}")
 print(f" Размер патча:
{cnn_config['patch_size']}x{cnn_config['patch_size']}")
 print(f" Скрытые каналы:
{cnn_config['hidden_channels']}")
 print(f" Количество классов: {cnn_config['num_classes']}")
 print(f" Dropout: {cnn_config['dropout']}")
 print(f" Batch Normalization:
{cnn_config['use_batch_norm']}")
 print(f" Активация: {cnn_config['activation']}")
 if config.USE_IMPROVED_CNN:
 print(f" Attention: {cnn_config.get('use_attention',
False)}")
 print()

 model = CNN(**cnn_config)
 num_params = sum(p.numel() for p in model.parameters())

```

```

print(f"Параметров модели: {num_params:,}")
print()

Создание папки для моделей
os.makedirs(config.MODELS_DIR, exist_ok=True)

Обучение
print("=" * 70)
print("НАЧАЛО ОБУЧЕНИЯ")
print("=" * 70)
print()

history = train_cnn_model(
 model=model,
 train_loader=train_loader,
 val_loader=val_loader,
 num_epochs=config.NUM_EPOCHS,
 learning_rate=config.LEARNING_RATE,
 device=device,
 models_dir=config.MODELS_DIR,
 class_weights=class_weights,
 use_focal_loss=config.USE_FOCAL_LOSS
)

Оценка на тестовых данных
print("=" * 70)
print("ОЦЕНКА НА ТЕСТОВЫХ ДАННЫХ")
print("=" * 70)
print()

Загружаем лучшую модель
best_model_path = os.path.join(config.MODELS_DIR,
config.BEST_MODEL_NAME)
model.load_state_dict(torch.load(best_model_path))

accuracy, preds, labels, cm = evaluate_model(model,
test_loader, device)

```

```

Сохраняем финальную модель
final_model_path = os.path.join(config.MODELS_DIR,
config.FINAL_MODEL_NAME)
torch.save(model.state_dict(), final_model_path)
print(f'Финальная модель сохранена:
{config.FINAL_MODEL_NAME}')
print()

print("=" * 70)
print("ОБУЧЕНИЕ ЗАВЕРШЕНО")
print("=" * 70)
print()
print(f'Результаты:')
print(f' Точность на тестовых данных: {accuracy:.4f}
({accuracy*100:.2f}%)')
print(f' Лучшая модель:
{config.BEST_MODEL_NAME}')
print(f' Финальная модель:
{config.FINAL_MODEL_NAME}')
print()
print("Модели сохранены в папке:", config.MODELS_DIR)

if __name__ == "__main__":
 main()

```

После того как вы вставили скрипт в файл .txt, сохраните изменения и измените формат на Python File (.py).

1: Откройте PowerShell и перейдите в папку CNN:

```

```powershell
cd <ПУТЬ_К_ПАПКЕ_CNN>
```

```

2: Запустите скрипт обучения:

```

```powershell
python scripts\02_train_cnn_model.py
```

```

### Шаг - 5.3. Что происходит во время обучения

#### Этап 1: Загрузка данных

...

Загрузка выборки из ...training\_samples\_expanded.csv...

[OK] Загружено 12,500 точек

Классы: [1, 2, 3, 4, 5]

Распределение по классам:

class

1 2500

2 2500

3 2500

4 2500

5 2500

...

#### Этап 2: Предобработка

...

Загрузка спутниковых данных...

[OK] Спутниковые данные: (6, 3471, 6921)

Каналов: 6

Размер: 3471 x 6921

Фильтрация выборки по границам Буланды...

[OK] Shapefile загружен: 1 объектов

CRS: EPSG:4326

Вычисление вегетационных индексов...

[OK] Вычислено индексов: 5

Загрузка DEM данных...

[OK] DEM: (3457, 6921)

Загрузка ночных данных DMSP-OLS...

[OK] Ночные данные: (3457, 6921)

...

#### Этап 3: Создание датасета

...

Создание датасета...

Размер входного вектора: 117

Базовые признаки: 13 (каналы: 6, индексы: 5, DEM: 1,  
ночные: 1)

Контекстные признаки: 104 (окно 3x3)  
...

Этап 4: Разделение данных  
...

Разделение данных на train/val/test...  
Обучающих образцов: 8,750 (70.0%)  
Валидационных образцов: 1,875 (15.0%)  
Тестовых образцов: 1,875 (15.0%)  
...

Этап 5: Обучение модели  
...

Начало обучения...  
Всего эпох: 200  
Батчей в эпоху (train): 137  
Батчей в эпоху (val): 30

Эпоха 1/200:  
Learning Rate: 0.000500  
Обучение: Loss = 1.5234, Accuracy = 0.4523 (45.23%)  
Валидация: Loss = 1.4892, Accuracy = 0.4687 (46.87%)  
[Модель сохранена: best\_ann\_model.pth] (лучшая точность: 46.87%)

Эпоха 2/200:  
Learning Rate: 0.000500  
Обучение: Loss = 1.4123, Accuracy = 0.5234 (52.34%)  
Валидация: Loss = 1.3891, Accuracy = 0.5412 (54.12%)  
[Модель сохранена: best\_ann\_model.pth] (лучшая точность: 54.12%)

...

Этап 6: Оценка на тестовых данных  
...

Оценка на тестовых данных...  
Оценка:

100% 

Точность на тестовых данных: 0.7523 (75.23%)

Матрица ошибок (Confusion Matrix):

| Предсказано ->   | Класс 1 | Класс 2 | Класс 3 | Класс 4 | Класс 5 |
|------------------|---------|---------|---------|---------|---------|
| Истинный класс 1 | 285     | 45      | 8       | 2       | 30      |
| Истинный класс 2 | 38      | 268     | 32      | 5       | 27      |
| Истинный класс 3 | 12      | 42      | 285     | 3       | 28      |
| Истинный класс 4 | 5       | 28      | 2       | 310     | 25      |
| Истинный класс 5 | 18      | 35      | 15      | 4       | 296     |

Отчет о классификации:

|                        | precision | recall | f1-score | support  |
|------------------------|-----------|--------|----------|----------|
| Пашня                  | 0.79      | 0.77   | 0.78     | 370      |
| Пастбища               | 0.64      | 0.72   | 0.68     | 370      |
| Леса                   | 0.83      | 0.77   | 0.80     | 369      |
| Вода                   | 0.96      | 0.84   | 0.90     | 370      |
| Застроенные территории |           | 0.73   | 0.80     | 0.76 368 |
| accuracy               | 0.75      | 1847   |          |          |
| macro avg              | 0.79      | 0.78   | 0.78     | 1847     |
| weighted avg           | 0.79      | 0.75   | 0.77     | 1847     |

Финальная модель сохранена: ann\_final\_model.pth

...

#### Шаг - 5.4. Результаты обучения

После завершения обучения в папке `results/models/` будут сохранены:

- `best\_cnn\_model.pth` - лучшая модель по валидационной точности
- `cnn\_final\_model.pth` - финальная модель после всех эпох

#### Шаг - 5.5. Интерпретация результатов

Точность на тестовых данных:

- > 80% - отличная точность
- 70-80% - хорошая точность
- 60-70% - приемлемая точность
- < 60% - низкая точность, нужны улучшения

Матрица ошибок:

- Показывает, какие классы путаются между собой
- Диагональ - правильные предсказания
- Вне диагонали - ошибки классификации

Отчет о классификации:

- precision - точность предсказания класса
- recall - полнота (сколько примеров класса найдено)
- f1-score - гармоническое среднее precision и recall

**Шаг - 5.6. Что делать, если точность низкая?**

Если точность < 60%:

1. Увеличьте LEARNING\_RATE:

```
```python
LEARNING_RATE = 0.001 # Было 0.0005
```
```

2. Увеличьте размеры сети:

```
```python
'hidden_channels: [1024, 512, 256, 128, 64] # Было [512,
256, 128, 64]
```
```

3. Уменьшите dropout:

```
```python
'dropout': 0.1 # Было 0.2
```
```

4. Увеличьте количество эпох:

```
```python
NUM_EPOCHS = 300 # Было 200
```
```

5. Проверьте качество обучающей выборки:

- Убедитесь, что точки правильно размечены
- Убедитесь, что все классы представлены достаточно

Если точность 60-70%:

1. Увеличьте CONTEXT\_WINDOW\_SIZE:

```
```python
CONTEXT_WINDOW_SIZE = 5 # Было 3
```
```

'''

2. Обучите ensemble (см. раздел 6)

3. Улучшите обучающую выборку:

- Добавьте больше точек для проблемных классов
- Используйте расширенную выборку (2500+ точек на

класс)

## Шаг - 6. ОБУЧЕНИЕ ENSEMBLE МОДЕЛЕЙ

### Шаг - 6.1. Что такое Ensemble?

Ensemble - это комбинация нескольких моделей для получения более точных предсказаний. Обычно дает прирост точности на 2-5% по сравнению с одиночной моделью.

### Шаг - 6.2. Как работает Ensemble?

1. Обучается несколько моделей (обычно 3-5) с разными начальными весами
2. Каждая модель делает свое предсказание
3. Финальное предсказание - это голосование большинством (majority voting)

### Шаг - 6.3. Когда использовать Ensemble?

- После получения хорошей базовой модели (70%+)
- Когда нужна максимальная точность
- Когда есть время на обучение нескольких моделей

### Шаг - 6.4. Настройка параметров Ensemble

Расположение: `config.py`, строка ~38

```python

USE_ENSEMBLE = False # Использовать ли ensemble из нескольких моделей

ENSEMBLE_SIZE = 3 # Количество моделей в ensemble

'''

Как изменить:

1. Откройте `config.py`
2. Найдите строки `USE\_ENSEMBLE` и `ENSEMBLE\_SIZE`
3. Измените значения (для обучения ensemble эти параметры не используются напрямую)

Рекомендации:

- ENSEMBLE_SIZE = 3 - стандартное значение (баланс точности и времени)

- ENSEMBLE_SIZE = 5 - для максимальной точности (больше времени обучения)

Шаг - 6.5. Запуск обучения Ensemble

Создайте файл формата .txt в папке scripts и назовите его 03_train_ensemble. Далее откройте файл через «Блокнот» и вставьте скрипт:

```
"""Обучение ensemble из нескольких MLP моделей для
улучшения точности"""
import sys
sys.stdout.reconfigure(encoding='utf-8')

import os
import torch
import torch.nn as nn
import numpy as np
from pathlib import Path
import random
import json

# Добавляем родительскую директорию в путь для импорта
parent_dir =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
sys.path.insert(0, parent_dir)
import config
from models.cnn import CNN
# Импорт utils из той же папки
from utils import LULCDataset, load_training_samples

# Импортируем функцию обучения из основного скрипта
import importlib.util
spec = importlib.util.spec_from_file_location("train_module",
os.path.join(os.path.dirname(__file__), "02_train_cnn_model.py"))
train_module = importlib.util.module_from_spec(spec)
spec.loader.exec_module(train_module)
train_cnn_model = train_module.train_cnn_model
evaluate_model = train_module.evaluate_model
from torch.utils.data import DataLoader,
WeightedRandomSampler
from sklearn.model_selection import train_test_split
```

```

from sklearn.metrics import accuracy_score

def
train_ensemble(num_models=config.NUM_ENSEMBLE_MODELS
, random_seeds=None):
    """Обучает ensemble из нескольких CNN моделей"""
    print("=" * 70)
    print(f"ОБУЧЕНИЕ ENSEMBLE ИЗ {num_models} CNN
МОДЕЛЕЙ")
    print("=" * 70)
    print()

    if random_seeds is None:
        random_seeds = [random.randint(0, 10000) for _ in
range(num_models)]

    device = torch.device(config.DEVICE)
    print(f"Устройство: {device}")
    print()

    # Загрузка данных
    satellite_path = config.get_satellite_path()
    training_samples_path = config.get_training_samples_path()
    dem_path = config.get_dem_path()
    nightlights_path = config.get_nightlights_path()

    samples_df = load_training_samples(training_samples_path)

    # Загрузка границ Буланды (если указаны)
    boundaries_path = config.get_boundaries_path()
    if boundaries_path:
        print(f" [OK] Используются границы Буланды:
{Path(boundaries_path).name}")
    else:
        print(f" [INFO] Границы Буланды не указаны (можно
указать BOUNDARIES_FILE в config.py)")
    print()

```

```

# Создание датасета
dataset = LULCDataset(
    satellite_path=satellite_path,
    samples_df=samples_df,
    dem_path=dem_path,
    nightlights_path=nightlights_path, # Ночные данные
DMSP-OLS
    normalize=True,
    augment=config.AUGMENT_DATA,
    patch_size=config.CNN_CONFIG['patch_size'], # Размер
патча для CNN
    boundaries_path=boundaries_path # Фильтрация по
границам Буланды
)

# Определяем количество входных каналов
sample_patch, _ = dataset[0]
input_channels = sample_patch.shape[0]
print(f'Количество входных каналов для ensemble:
{input_channels}')
print(f'Размер патча:
{config.CNN_CONFIG['patch_size']} x {config.CNN_CONFIG['patch
size']}')
print()

# Разделение на train/val/test (одинаковое для всех
моделей)
indices = np.arange(len(dataset))
train_indices, temp_indices = train_test_split(
    indices,
    test_size=(config.VAL_RATIO + config.TEST_RATIO),
    random_state=42, # Фиксированный сид для разделения
данных
    stratify=dataset.samples_df['class'].values
)
val_indices, test_indices = train_test_split(
    temp_indices,

```

```

        test_size=(config.TEST_RATIO / (config.VAL_RATIO +
config.TEST_RATIO)),
        random_state=42, # Фиксированный сид для разделения
данных

```

```

stratify=dataset.samples_df.iloc[temp_indices]['class'].values
)

```

```

# Вычисляем веса классов для балансировки loss
class_weights = None
if config.USE_CLASS_WEIGHTS:
    print("Вычисление весов классов для ensemble...")
    train_classes =
dataset.samples_df.iloc[train_indices]['class'].values
    unique_classes, class_counts = np.unique(train_classes,
return_counts=True)
    total_samples = len(train_classes)
    num_classes = len(unique_classes)

```

```

    class_weights_dict = {}
    for cls, count in zip(unique_classes, class_counts):
        class_weights_dict[int(cls)] = total_samples /
(num_classes * count)

```

```

    class_weights = torch.ones(6)
    for cls, weight in class_weights_dict.items():
        if 0 <= cls < 6:
            class_weights[cls] = weight

    class_weights = class_weights / class_weights.mean()
    print(f" Веса классов: {dict(zip(range(6),
class_weights.numpy()))}")
    print()

```

```

# Oversampling
if config.USE_OVERSAMPLING:
    train_classes =
dataset.samples_df.iloc[train_indices]['class'].values

```

```

class_counts = np.bincount(train_classes, minlength=6)
problem_classes = [2, 5]
target_count = int(np.max(class_counts[1:]))

oversampled_indices = list(train_indices)
for cls in problem_classes:
    cls_mask = train_classes == cls
    cls_indices = train_indices[cls_mask]
    current_count = len(cls_indices)
    if current_count < target_count:
        needed = target_count - current_count
        oversample_indices = np.random.choice(cls_indices,
size=needed, replace=True)
        oversampled_indices.extend(oversample_indices)
train_indices = np.array(oversampled_indices)

train_dataset = torch.utils.data.Subset(dataset, train_indices)
val_dataset = torch.utils.data.Subset(dataset, val_indices)
test_dataset = torch.utils.data.Subset(dataset, test_indices)

# Weighted Random Sampler для балансировки классов в
обучении
train_labels =
dataset.samples_df.iloc[train_indices]['class'].values
unique_train_classes, counts_train_classes =
np.unique(train_labels, return_counts=True)
min_count = counts_train_classes.min()
max_count = counts_train_classes.max()

if hasattr(config, 'USE_WEIGHTED_SAMPLER') and
config.USE_WEIGHTED_SAMPLER and (max_count / min_count
> 1.5):
    print("Используется WeightedRandomSampler для
балансировки батчей ensemble...")
    class_sample_counts = np.array([len(np.where(train_labels
== t)[0]) for t in unique_train_classes])
    weight = 1. / class_sample_counts

```

```

        samples_weight =
np.array([weight[np.where(unique_train_classes == t)[0][0]] for t in
train_labels])
        samples_weight =
torch.from_numpy(samples_weight).double()

        sampler = WeightedRandomSampler(samples_weight,
len(samples_weight))
        train_loader = DataLoader(train_dataset,
batch_size=config.BATCH_SIZE, sampler=sampler,
num_workers=0, pin_memory=True)
    else:
        print("Классы сбалансированы или
WeightedRandomSampler отключен. Используется обычный
shuffle для ensemble.")
        train_loader = DataLoader(train_dataset,
batch_size=config.BATCH_SIZE, shuffle=True, num_workers=0,
pin_memory=True)

        val_loader = DataLoader(val_dataset,
batch_size=config.BATCH_SIZE, shuffle=False, num_workers=0,
pin_memory=True)
        test_loader = DataLoader(test_dataset,
batch_size=config.BATCH_SIZE, shuffle=False, num_workers=0,
pin_memory=True)

        print(f"Обучающих образцов ensemble:
{len(train_dataset):,} ( {len(train_dataset)/len(dataset)*100:.1f}%)")
        print(f"Валидационных образцов ensemble:
{len(val_dataset):,} ( {len(val_dataset)/len(dataset)*100:.1f}%)")
        print(f"Тестовых образцов ensemble: {len(test_dataset):,}
( {len(test_dataset)/len(dataset)*100:.1f}%)")
        print(f"Батчей в обучающей выборке ensemble за эпоху:
{len(train_loader)}")
        print()

        models = []
        model_paths = []

```

```

os.makedirs(config.MODELS_DIR, exist_ok=True)

for i, seed in enumerate(random_seeds):
    print(f'\n{'='*70}')
    print(f'ОБУЧЕНИЕ CNN МОДЕЛИ
{i+1}/{num_models} (SEED: {seed})')
    print(f'{'='*70}')

    # Устанавливаем сид для воспроизводимости
    torch.manual_seed(seed)
    np.random.seed(seed)
    random.seed(seed)
    if device.type == 'cuda':
        torch.cuda.manual_seed(seed)
        torch.backends.cudnn.deterministic = True
        torch.backends.cudnn.benchmark = False

    cnn_config = config.CNN_CONFIG.copy()
    cnn_config['input_channels'] = input_channels
    model = CNN(**cnn_config)

    history = train_cnn_model(
        model=model,
        train_loader=train_loader,
        val_loader=val_loader,
        num_epochs=config.NUM_EPOCHS,
        learning_rate=config.LEARNING_RATE,
        device=device,
        models_dir=config.MODELS_DIR,
        class_weights=class_weights,
        use_focal_loss=config.USE_FOCAL_LOSS
    )

    # Загружаем лучшую модель для оценки и сохранения
    best_model_path_single =
os.path.join(config.MODELS_DIR, config.BEST_MODEL_NAME)
    model.load_state_dict(torch.load(best_model_path_single))

```

```

        # Сохраняем модель ensemble с уникальным именем
        ensemble_model_name =
f"ensemble_cnn_model_{i+1}_seed{seed}.pth"
        ensemble_model_path =
os.path.join(config.MODELS_DIR, ensemble_model_name)
        torch.save(model.state_dict(), ensemble_model_path)
        print(f"Модель {i+1} сохранена:
{ensemble_model_name}")

        models.append(model)
        model_paths.append(ensemble_model_path)
        print(f"Точность модели {i+1} на валидации:
{history['best_val_acc']*100:.2f}%")
        print()

    print("=" * 70)
    print("ОЦЕНКА ENSEMBLE НА ТЕСТОВЫХ
ДААННЫХ")
    print("=" * 70)

    all_preds = []
    all_labels = []

    for model_idx, model in enumerate(models):
        model.eval()
        model_preds = []
        model_labels = []

        with torch.no_grad():
            for batch in test_loader:
                inputs, labels = batch
                inputs = inputs.to(device)
                labels = labels.to(device)

                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                model_preds.extend(preds.cpu().numpy())

```

```

        model_labels.extend(labels.cpu().numpy())

    all_preds.append(model_preds)
    if len(all_labels) == 0:
        all_labels = model_labels

# Ensemble предсказания (голосование большинством)
ensemble_preds = []
for i in range(len(all_labels)):
    votes = [preds[i] for preds in all_preds]
    # Используем Counter для более надежного majority
voting
    from collections import Counter
    ensemble_pred = Counter(votes).most_common(1)[0][0]
    ensemble_preds.append(ensemble_pred)

# Вычисляем точность
accuracy = accuracy_score(all_labels, ensemble_preds)
print(f"Точность ensemble ({num_models} моделей):
{accuracy:.4f} ( {accuracy*100:.2f}%)")
print()

# Сохраняем информацию об ensemble
# Создаем снимок конфига только с сериализуемыми
значениями
config_snapshot = {}
for k in dir(config):
    if k.startswith('__') or callable(getattr(config, k)):
        continue
    try:
        value = getattr(config, k)
        # Проверяем, можно ли сериализовать
        if isinstance(value, (str, int, float, bool, list, dict,
type(None))):
            # Для списков и словарей проверяем вложенные
значения
            if isinstance(value, list):
                if all(isinstance(x, (str, int, float, bool)) for x in value):

```

```

        config_snapshot[k] = value
        elif isinstance(value, dict):
            if all(isinstance(kk, (str, int, float, bool)) and isinstance(vv, (str,
int, float, bool, list, dict, type(None))) for kk, vv in value.items()):
                config_snapshot[k] = value
            else:
                config_snapshot[k] = value
        except:
            pass # Пропускаем значения, которые нельзя

```

получить

```

ensemble_info = {
    'num_models': num_models,
    'model_paths': model_paths,
    'test_accuracy': float(accuracy),
    'random_seeds': random_seeds,
    'config_snapshot': config_snapshot
}

info_path = os.path.join(config.MODELS_DIR,
'ensemble_info.json')
with open(info_path, 'w', encoding='utf-8') as f:
    json.dump(ensemble_info, f, indent=2, ensure_ascii=False)

print(f"Информация об ensemble сохранена: {info_path}")
print()
print("=" * 70)
print("ENSEMBLE ОБУЧЕН")
print("=" * 70)
print(f"Точность ensemble: {accuracy*100:.2f}%")
print(f"Модели сохранены в: {config.MODELS_DIR}")

if __name__ == "__main__":

train_ensemble(num_models=config.NUM_ENSEMBLE_MODELS
)

```

После того как вы вставили скрипт в файл .txt, сохраните изменения и измените формат на Python File (.py).

Шаг 1: Убедитесь, что базовая модель обучена:

```
'''  
cd <ПУТЬ_К_ПАПКЕ_CNN>  
dir results\models\best_cnn_model.pth  
'''
```

Шаг 2: Запустите скрипт обучения ensemble:

```
'''  
python scripts\03_train_ensemble.py  
'''
```

Шаг - 6.6. Что происходит во время обучения Ensemble

Этап 1: Обучение первой модели

```
'''
```

ОБУЧЕНИЕ ENSEMBLE ИЗ 3 МОДЕЛЕЙ

```
=====
```

```
=====
```

Обучение модели 1/3...

Seed: 42

[Обучение модели...]

...

Точность модели 1: 75.23%

```
'''
```

Этап 2: Обучение второй модели

```
'''
```

Обучение модели 2/3...

Seed: 1234

[Обучение модели...]

...

Точность модели 2: 74.87%

```
'''
```

Этап 3: Обучение третьей модели

```
'''
```

Обучение модели 3/3...

Seed: 5678

[Обучение модели...]

...

Точность модели 3: 75.45%

```
'''
```

Этап 4: Оценка Ensemble

```
'''
```

ОЦЕНКА ENSEMBLE НА ТЕСТОВЫХ ДАННЫХ

```
Точность ensemble (3 моделей): 0.7689 (76.89%)
```

```
'''
```

Шаг - 6.7. Результаты обучения Ensemble

После завершения обучения в папке `results/models/` будут сохранены:

- `ensemble\_model\_1.pth` - первая модель ensemble
- `ensemble\_model\_2.pth` - вторая модель ensemble
- `ensemble\_model\_3.pth` - третья модель ensemble
- `ensemble\_info.json` - информация об ensemble (пути к

моделям, точность)

Шаг - 6.8. Настройка количества моделей в Ensemble

Если нужно изменить количество моделей, откройте `scripts/03\_train\_ensemble.py` и измените вызов функции:

```
'''python
if __name__ == "__main__":
    train_ensemble(num_models=5) # Было 3
'''
```

Или передайте параметр через командную строку (если скрипт поддерживает).

Шаг - 7. КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЯ

Шаг - 7.1. Подготовка к классификации

1: Убедитесь, что модель обучена:

```
'''
```

```
cd <ПУТЬ_К_ПАПКЕ_CNN>
dir results\models\best_cnn_model.pth
'''
```

Или для ensemble:

```
'''
```

```
dir results\models\ensemble_info.json
'''
```

Шаг - 7.2. Запуск классификации

Создайте файл формата .txt в папке scripts и назовите его 04_classify_image. Далее откройте файл через «Блокнот» и вставьте скрипт:

```
"""
Классификация всего изображения с использованием
обученной CNN модели или ensemble
Этот скрипт выполняет полный цикл классификации:
1. Загрузка модели (одиночной или ensemble)
2. Классификация всего изображения
3. Сохранение результата в формате GeoTIFF
4. Создание PNG визуализации с картой и статистикой
5. Оценка точности по классам
6. Автоматическое открытие PNG изображения
Запуск: python scripts/04_classify_image.py
"""

import sys
sys.stdout.reconfigure(encoding='utf-8')

import os
import torch
import numpy as np
import rasterio
from tqdm import tqdm
from pathlib import Path
import json
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.colors import ListedColormap, BoundaryNorm

# Добавляем родительскую директорию в путь для импорта
parent_dir =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
sys.path.insert(0, parent_dir)
import config
# Используем улучшенную модель, так как модель была
обучена с ImprovedCNN
if config.USE_IMPROVED_CNN:
    from models.cnn_improved import ImprovedCNN as CNN
```

```

else:
    from models.cnn import CNN
    # Импорт utils из той же папки
    from utils import LULCDataset

    def create_classification_visualization(classification_array,
class_stats, output_path, model_architecture="CNN",
is_ensemble=False, accuracy_stats=None):
        """
        Создает PNG визуализацию классификации с легендой и
        статистикой

        Args:
            classification_array: массив классификации (numpy
array)
            class_stats: словарь со статистикой по классам
            output_path: путь для сохранения PNG
            model_architecture: название архитектуры модели
(CNN, MLP, ANN)
            is_ensemble: используется ли ensemble
            accuracy_stats: словарь с точностью по классам
(опционально)
        """
        # Цветовая схема для классов LULC (только для карты)
        class_colors = {
            0: '#FFFFFF', # Фон - белый
            1: '#FFFF00', # Пашня - желтый
            2: '#00FF00', # Пастбища - зеленый
            3: '#006400', # Леса - темно-зеленый
            4: '#0000FF', # Вода - синий
            5: '#FF0000' # Застроенные территории - красный
        }

        # Создаем colormap
        colors = [class_colors.get(i, '#808080') for i in range(6)]
        cmap = ListedColormap(colors)
        bounds = [0, 1, 2, 3, 4, 5, 6]
        norm = BoundaryNorm(bounds, cmap.N)

```

```

# Создаем фигуру с двумя областями: карта и статистика
fig = plt.figure(figsize=(20, 12), dpi=150)
gs = fig.add_gridspec(1, 2, width_ratios=[3, 1], hspace=0.1,
wspace=0.05)

# Левая часть - карта классификации
ax_map = fig.add_subplot(gs[0])
im = ax_map.imshow(classification_array, cmap=cmap,
norm=norm,
interpolation='nearest', origin='upper')

# Заголовок с указанием архитектуры модели
if is_ensemble:
    title = f'Классификация LULC - {model_architecture}'
Ensemble модель'
else:
    title = f'Классификация LULC - {model_architecture}'
модель'
ax_map.set_title(title, fontsize=20, fontweight='bold',
pad=20)

# Убираем оси
ax_map.set_xticks([])
ax_map.set_yticks([])
ax_map.axis('off')

# Правая часть - статистика
ax_stats = fig.add_subplot(gs[1])
ax_stats.axis('off')

# Заголовок статистики
ax_stats.text(0.1, 0.95, 'СТАТИСТИКА
КЛАССИФИКАЦИИ',
transform=ax_stats.transAxes, fontsize=16,
fontweight='bold', verticalalignment='top')

y_pos = 0.88

```

```

# Распределение пикселей по классам
ax_stats.text(0.1, y_pos, 'Распределение по классам:',
              transform=ax_stats.transAxes, fontsize=12,
              fontweight='bold', verticalalignment='top')
y_pos -= 0.05

for class_id in sorted(class_stats.keys()):
    if class_id == 0: # Пропускаем фон
        continue
    stats = class_stats[class_id]
    # Черный текст для распределения классов (без
цветовой подсветки)
    text = f"{stats['name']}: {stats['percentage']:.1f}%"
    text += f"\n ({stats['count']:,} пикс.)"
    ax_stats.text(0.1, y_pos, text,
transform=ax_stats.transAxes,
    fontsize=10, color='black', fontweight='bold',
    verticalalignment='top')
    y_pos -= 0.08

# Точность по классам (если есть)
if accuracy_stats and 'class_accuracy' in accuracy_stats:
    y_pos -= 0.05
    ax_stats.text(0.1, y_pos, 'Точность по классам:',
transform=ax_stats.transAxes, fontsize=12,
    fontweight='bold', verticalalignment='top')
    y_pos -= 0.05

    for class_id in
sorted(accuracy_stats['class_accuracy'].keys()):
        if class_id == 0: # Пропускаем фон
            continue
        acc_data = accuracy_stats['class_accuracy'][class_id]
        accuracy = acc_data['accuracy']
        correct = acc_data['correct']
        total = acc_data['total']

```

```

class_name = config.LULC_CLASSES.get(class_id,
fКласс {class_id}')

# Цвет в зависимости от точности
if accuracy >= 80:
    acc_color = 'green'
elif accuracy >= 60:
    acc_color = 'orange'
else:
    acc_color = 'red'

text = f'{class_name}: {accuracy:.1f}%'
text += f'\n ({correct}/{total})'
ax_stats.text(0.1, y_pos, text,
transform=ax_stats.transAxes,
    fontsize=10, color=acc_color, fontweight='bold',
    verticalalignment='top')
    y_pos -= 0.08

# Общая точность
if 'overall_accuracy' in accuracy_stats:
    y_pos -= 0.05
    overall_acc = accuracy_stats['overall_accuracy']
    overall_total = accuracy_stats['overall_total']
    overall_correct = accuracy_stats['overall_correct']

    acc_color = 'green' if overall_acc >= 70 else 'orange' if
overall_acc >= 50 else 'red'

    ax_stats.text(0.1, y_pos, 'Общая точность:',
transform=ax_stats.transAxes, fontsize=12,
    fontweight='bold', verticalalignment='top')
    y_pos -= 0.05
    text = f'{overall_acc:.2f}%'
    text += f'\n ({overall_correct}/{overall_total})'
    ax_stats.text(0.1, y_pos, text,
transform=ax_stats.transAxes,
    fontsize=14, color=acc_color, fontweight='bold',

```

```

verticalalignment='top')

plt.tight_layout()

# Сохраняем
os.makedirs(os.path.dirname(output_path), exist_ok=True)
plt.savefig(output_path, dpi=150, bbox_inches='tight',
pad_inches=0.2)
plt.close()

# Автоматически открываем изображение
try:
    if os.name == 'nt': # Windows
        os.startfile(output_path)
    elif os.name == 'posix': # macOS и Linux
        import subprocess
        subprocess.run(['open' if sys.platform == 'darwin' else
'xdg-open', output_path])
    print(f"[OK] Изображение открыто автоматически")
except Exception as e:
    print(f"[INFO] Не удалось открыть изображение
автоматически: {e}")

def evaluate_classification_accuracy(classification_array,
satellite_path, class_stats):
    """
    Оценивает точность классификации по классам на основе
    обучающей выборки

    Args:
        classification_array: массив классификации
        satellite_path: путь к спутниковым данным (для
получения координат)
        class_stats: статистика по классам

    Returns:

```

dict: словарь с данными о точности для отображения на изображении

```
"""
accuracy_stats = None
try:
    from utils import load_training_samples
    import rasterio.transform

    # Загружаем обучающую выборку
    samples_path = config.get_training_samples_path()
    samples_df = load_training_samples(samples_path)

    # Загружаем transform из спутниковых данных
    with rasterio.open(satellite_path) as src:
        transform = src.transform

    # Подсчитываем совпадения
    correct_by_class = {cls: 0 for cls in class_stats.keys() if cls
> 0}
    total_by_class = {cls: 0 for cls in class_stats.keys() if cls >
0}

    for idx, row in samples_df.iterrows():
        lon = row['x']
        lat = row['y']
        true_class = int(row['class'])

        if true_class == 0: # Пропускаем фон
            continue

        # Преобразуем координаты в индексы пикселей
        row_idx, col_idx = rasterio.transform.rowcol(transform,
lon, lat)

        # Проверяем границы
        if 0 <= row_idx < classification_array.shape[0] and 0 <=
col_idx < classification_array.shape[1]:
```

```

        pred_class = int(classification_array[row_idx,
col_idx])
        total_by_class[true_class] =
total_by_class.get(true_class, 0) + 1
        if pred_class == true_class:
            correct_by_class[true_class] = correct_by_class.get(true_class,
0) + 1

# Формируем данные о точности
class_accuracy = {}
overall_correct = 0
overall_total = 0

for class_id in sorted(correct_by_class.keys()):
    correct = correct_by_class[class_id]
    total = total_by_class[class_id]
    if total > 0:
        accuracy = correct / total * 100
        overall_correct += correct
        overall_total += total
        class_accuracy[class_id] = {
'accuracy': accuracy,
'correct': correct,
'total': total
        }

    if overall_total > 0:
        overall_accuracy = overall_correct / overall_total * 100
        accuracy_stats = {
            'class_accuracy': class_accuracy,
            'overall_accuracy': overall_accuracy,
            'overall_correct': overall_correct,
            'overall_total': overall_total
        }

except Exception as e:
    print(f" [INFO] Оценка точности пропущена: {e}")

```

```

return accuracy_stats

def extract_patch_for_pixel(satellite_data, indices_data,
dem_data, nightlights_data, row, col,
    satellite_mean, satellite_std, indices_mean, indices_std,
    dem_mean, dem_std, nightlights_mean, nightlights_std,
    patch_size=7, normalize=True):
    """Извлекает патч для одного пикселя (для CNN)"""
    half_patch = patch_size // 2

def extract_patch(data_array, row_idx, col_idx):
    """Извлекает патч с padding для граничных случаев"""
    row_start = max(0, row_idx - half_patch)
    row_end = min(data_array.shape[0], row_idx + half_patch
+ 1)
    col_start = max(0, col_idx - half_patch)
    col_end = min(data_array.shape[1], col_idx + half_patch +
1)

    patch = data_array[row_start:row_end,
col_start:col_end].copy()

    # Padding если патч меньше нужного размера
    if patch.shape[0] < patch_size or patch.shape[1] <
patch_size:
        padded_patch = np.zeros((patch_size, patch_size),
dtype=patch.dtype)
        pad_row_start = half_patch - (row_idx - row_start)
        pad_row_end = pad_row_start + patch.shape[0]
        pad_col_start = half_patch - (col_idx - col_start)
        pad_col_end = pad_col_start + patch.shape[1]
        padded_patch[pad_row_start:pad_row_end,
pad_col_start:pad_col_end] = patch
        patch = padded_patch

    return patch

```

```

channels = []

# Спутниковые каналы
for ch in range(satellite_data.shape[0]):
    patch = extract_patch(satellite_data[ch], row, col)
    if normalize:
        patch = (patch - satellite_mean[ch]) / satellite_std[ch]
    patch = np.nan_to_num(patch, nan=0.0)
    patch[patch < -9999] = 0.0
    channels.append(patch)

# Вегетационные индексы
for index_name, index_data in indices_data.items():
    patch = extract_patch(index_data, row, col)
    if normalize:
        patch = (patch - indices_mean[index_name]) /
indices_std[index_name]
    patch = np.nan_to_num(patch, nan=0.0)
    channels.append(patch)

# DEM
if dem_data is not None:
    patch = extract_patch(dem_data, row, col)
    if normalize:
        patch = (patch - dem_mean) / dem_std
    patch = np.nan_to_num(patch, nan=0.0)
    patch[patch < -9999] = 0.0
    channels.append(patch)

# Ночные данные
if nightlights_data is not None:
    patch = extract_patch(nightlights_data, row, col)
    if normalize:
        patch = (patch - nightlights_mean) / nightlights_std
    patch = np.nan_to_num(patch, nan=0.0)
    patch[patch < -9999] = 0.0
    channels.append(patch)

```

```

        # Объединяем все каналы в один тензор (channels,
patch_size, patch_size)
        patch_tensor = np.stack(channels, axis=0).astype(np.float32)

        return patch_tensor

def extract_context_features(row_idx, col_idx, data_array,
mean_val, std_val, context_window):
    """Извлекает контекстные признаки из окна"""
    half_window = context_window // 2

    row_start = max(0, row_idx - half_window)
    row_end = min(data_array.shape[0], row_idx + half_window
+ 1)
    col_start = max(0, col_idx - half_window)
    col_end = min(data_array.shape[1], col_idx + half_window +
1)

    window = data_array[row_start:row_end,
col_start:col_end].copy()

    valid_mask = ~np.isnan(window) & (window > -9999)
    if np.any(valid_mask):
        valid_data = window[valid_mask]
        mean = np.mean(valid_data)
        std = np.std(valid_data) if len(valid_data) > 1 else 0.0
        min_val = np.min(valid_data)
        max_val = np.max(valid_data)
        median = np.median(valid_data)

        if window.shape[0] >= 2 and window.shape[1] >= 2:
            grad_x = np.abs(np.diff(window, axis=1)).mean() if
window.shape[1] > 1 else 0.0
            grad_y = np.abs(np.diff(window, axis=0)).mean() if
window.shape[0] > 1 else 0.0
            total_grad = np.sqrt(grad_x**2 + grad_y**2) if grad_x >
0 or grad_y > 0 else 0.0

```

```

        else:
            grad_x = grad_y = total_grad = 0.0
        else:
            mean = std = min_val = max_val = median = grad_x =
grad_y = total_grad = 0.0

        if mean_val != 0 or std_val != 1:
            mean = (mean - mean_val) / std_val if std_val > 0 else 0.0
            std = std / std_val if std_val > 0 else 0.0
            median = (median - mean_val) / std_val if std_val > 0 else
0.0

    return [mean, std, min_val, max_val, median, grad_x, grad_y,
total_grad]

```

```

def classify_image_ensemble(models, satellite_path, dem_path,
nightlights_path, output_path, device, batch_size=1000,
is_ensemble=False, boundaries_path=None):
    """

```

Классификация с использованием ensemble моделей или
одиночной модели

Args:

models: список моделей (для ensemble) или одна модель
satellite_path: путь к спутниковым данным
dem_path: путь к DEM
nightlights_path: путь к ночным данным DMSP-OLS
output_path: путь для сохранения результата
device: устройство (CPU/GPU)
batch_size: размер батча
is_ensemble: True если используется ensemble
boundaries_path: путь к shapfile с границами Буланды

(опционально)

"""

```

for model in models:
    model.to(device)
    model.eval()

```

```

        # Загружаем границы Буланды (если указаны)
        boundaries_mask = None
        if boundaries_path:
            print("Загрузка границ Буланды для маскирования...")
            from utils import load_boundaries_shapefile,
create_boundaries_mask
            boundaries_gdf =
load_boundaries_shapefile(boundaries_path)
            if boundaries_gdf is not None:
                mask_result = create_boundaries_mask(satellite_path,
boundaries_gdf)
                if mask_result is not None:
                    boundaries_mask, _, _, _ = mask_result
                    print(f" [OK] Маска границ создана")
                else:
                    print(f" [WARNING] Не удалось создать маску
границ")
            else:
                print(f" [WARNING] Shapefile не загружен")
        print()

        # Создаем временный датасет для получения статистики
нормализации
        from utils import load_training_samples
        import pandas as pd

        samples_path = config.get_training_samples_path()
        samples_df = load_training_samples(samples_path)
        # Берем только одну точку для инициализации
        temp_df = samples_df.head(1)

        temp_dataset = LULCDataset(
            satellite_path=satellite_path,
            samples_df=temp_df,
            dem_path=dem_path,
            nightlights_path=config.get_nightlights_path(), # Ночные
данные DMSP-OLS
            normalize=True,

```

```

        augment=False,
        patch_size=config.CNN_CONFIG['patch_size'],
        boundaries_path=boundaries_path # Фильтрация по
границам
    )

    # Получаем статистику и данные
    satellite_data = temp_dataset.satellite_data
    indices_data = temp_dataset.indices_data
    dem_data = temp_dataset.dem_data
    nightlights_data = temp_dataset.nightlights_data

    satellite_mean = temp_dataset.satellite_mean
    satellite_std = temp_dataset.satellite_std
    indices_mean = temp_dataset.indices_mean
    indices_std = temp_dataset.indices_std
    dem_mean = temp_dataset.dem_mean
    dem_std = temp_dataset.dem_std
    nightlights_mean = temp_dataset.nightlights_mean
    nightlights_std = temp_dataset.nightlights_std

    transform = temp_dataset.transform
    crs = temp_dataset.crs
    height = temp_dataset.height
    width = temp_dataset.width

    print(f" Размер изображения: {height} x {width}
пикселей")
    print(f" Каналов: {satellite_data.shape[0]}")
    print(f" Индексов: {len(indices_data)}")
    if dem_data is not None:
        print(f" DEM: включен")
    if nightlights_data is not None:
        print(f" Ночные данные DMSP-OLS: включены")
    print()

    # Создаем массив для результата
    result = np.zeros((height, width), dtype=np.uint8)

```

```

# Если есть маска границ, собираем все пиксели внутри
границ
    if boundaries_mask is not None:
        print(f" [INFO] Классификация будет выполняться
только внутри границ Буланды")
        print(f"      Пиксели вне границ будут помечены как
фон (класс 0)")
        # Создаем маску валидных пикселей (внутри границ)
        valid_mask = (boundaries_mask == 1).astype(bool)
        valid_coords = np.where(valid_mask)
        num_valid = len(valid_coords[0])
        print(f" [INFO] Валидных пикселей для
классификации: {num_valid:,}")
    else:
        # Если маски нет, обрабатываем все пиксели
        valid_coords = (np.arange(height).repeat(width),
np.tile(np.arange(width), height))
        num_valid = height * width
        print(f" [INFO] Обработка всех пикселей:
{num_valid:,}")
        print()

        print(f"Обработка пикселей батчами по {batch_size}...")
        print()

        with torch.no_grad():
            # Обрабатываем валидные пиксели батчами (как в
старом скрипте)
            for i in tqdm(range(0, num_valid, batch_size),
desc="Классификация"):
                end_idx = min(i + batch_size, num_valid)
                batch_rows = valid_coords[0][i:end_idx]
                batch_cols = valid_coords[1][i:end_idx]

                # Извлекаем патчи для батча
                batch_features = []
                batch_positions = []

```

```

        for row, col in zip(batch_rows, batch_cols):
            patch = extract_patch_for_pixel(
                satellite_data, indices_data, dem_data, nightlights_data, row,
col,
                satellite_mean, satellite_std, indices_mean, indices_std,
                dem_mean, dem_std, nightlights_mean, nightlights_std,
                patch_size=config.CNN_CONFIG['patch_size'],
                normalize=True
            )

```

```

            batch_features.append(patch)
            batch_positions.append((row, col))

```

```

        # Обработываем батч
        if len(batch_features) > 0:
            # Объединяем патчи в батч (batch, channels,
patch_size, patch_size)
            batch_tensor =
torch.FloatTensor(np.stack(batch_features, axis=0)).to(device)

```

```

            # Ensemble предсказания
            all_preds = []
            for model in models:
                outputs = model(batch_tensor)
                _, preds = torch.max(outputs, 1)
                all_preds.append(preds.cpu().numpy())

```

```

            # Majority voting с учетом ночных данных для
застроенных территорий
            ensemble_preds = []
            for idx in range(len(batch_features)):
                votes = [preds[idx] for preds in all_preds]
                ensemble_pred = max(set(votes), key=votes.count)

```

```

            # Усиление использования ночных данных для застроенных
территорий (с более строгими порогами)
            if nightlights_data is not None:

```

```

    r, c = batch_positions[idx]
    if 0 <= r < nightlights_data.shape[0] and 0 <= c <
nightlights_data.shape[1]:
        night_val = nightlights_data[r, c]
        # Используем более строгие пороги: только очень
яркие области (ядро света)
        # Порог увеличен с 0.5*std до 1.5*std для уменьшения
влияния рассеянного света
        if not np.isnan(night_val) and night_val > nightlights_mean
+ 1.5 * nightlights_std:
            # Только если модель уже предсказала класс 5,
подтверждаем его
            # Или если очень яркий свет (2.0*std) и предсказание
близко к застройке
            if ensemble_pred == 5:
                # Подтверждаем застроенную территорию только
для очень ярких областей
                pass # Оставляем как есть
            elif night_val > nightlights_mean + 2.0 * nightlights_std
and ensemble_pred in [1, 2]:
                # Только для очень ярких ядер света
переклассифицируем пашню/пастбища в застройку
                ensemble_pred = 5

ensemble_preds.append(ensemble_pred)

# Сохраняем результаты
for idx, (r, c) in enumerate(batch_positions):
result[r, c] = ensemble_preds[idx]

# Сохранение результата
print(f"\nСохранение результата в {output_path}...")
os.makedirs(os.path.dirname(output_path), exist_ok=True)

with rasterio.open(
    output_path,
    'w',
    driver='GTiff',

```

```

        height=height,
        width=width,
        count=1,
        dtype=result.dtype,
        crs=crs,
        transform=transform,
        compress='lzw'
    ) as dst:
        dst.write(result, 1)

    print(f"[OK] Результат сохранен: {output_path}")

    # Статистика по классам (только для области Буланды,
    без фона)
    print("\nСтатистика классификации (только область
    Буланды):")
    print("-" * 70)

    # Создаем маску области Буланды (исключаем фон -
    класс 0)
    if boundaries_mask is not None:
        bulanda_mask = (boundaries_mask == 1) & (result != 0)
    else:
        bulanda_mask = (result != 0)

    # Подсчитываем статистику только для пикселей внутри
    Буланды
    bulanda_pixels = result[bulanda_mask]
    unique_classes, counts = np.unique(bulanda_pixels,
    return_counts=True)
    total_pixels_bulanda = counts.sum()

    class_stats = {}
    for class_id, count in zip(unique_classes, counts):
        if int(class_id) == 0: # Пропускаем фон
            continue
        class_name = config.LULC_CLASSES.get(int(class_id),
    f'Класс {int(class_id)}')

```

```

        percentage = count / total_pixels_bulanda * 100
        class_stats[int(class_id)] = {'name': class_name, 'count':
count, 'percentage': percentage}
        print(f" Класс {int(class_id)} ({class_name}): {count:},
пикселей ({percentage:.2f}%)")
        print()

# Оценка точности по классам (сравнение с обучающей
выборкой)
print("Оценка точности по классам...")
accuracy_stats = evaluate_classification_accuracy(result,
satellite_path, class_stats)

# Создание PNG визуализации с отображением
статистики
print("Создание PNG визуализации...")
png_output_path = output_path.replace('.tif', '.png')
# Определяем архитектуру модели из названия файла или
конфигурации
model_architecture = "CNN" # Для CNN проекта
create_classification_visualization(result, class_stats,
png_output_path, model_architecture, is_ensemble, accuracy_stats)
print(f"[OK] Визуализация сохранена: {png_output_path}")
print()

def main():
    """Основная функция"""
    print("=" * 70)
    print("КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЯ С
ИСПОЛЬЗОВАНИЕМ CNN МОДЕЛИ/ENSEMBLE")
    print("=" * 70)
    print()

    device = torch.device(config.DEVICE)
    print(f"Устройство: {device}")
    print()

# Проверка наличия ensemble или одиночной модели

```

```

# Временно используем одиночную модель, так как она
обучена с расширенными данными
ensemble_info_path = os.path.join(config.MODELS_DIR,
'ensemble_info.json')
use_ensemble = False # Используем одиночную модель
(cnn_final_model.pth)

if use_ensemble:
    print("Используется ENSEMBLE моделей")
    # Загружаем информацию об ensemble
    with open(ensemble_info_path, 'r', encoding='utf-8') as f:
        ensemble_info = json.load(f)

    num_models = ensemble_info['num_models']
    model_paths = ensemble_info['model_paths']
    print(f" Количество моделей: {num_models}")
    print()

    # Загружаем все модели ensemble
    models = []
    for i, model_path in enumerate(model_paths):
        if not Path(model_path).exists():
            print(f"[ОШИБКА] Модель {i+1} не найдена:
{model_path}")
            return

    # Определяем размер входа из первой модели
    if i == 0:
        # Создаем временный датасет для определения
размера
        from utils import load_training_samples
        samples_path = config.get_training_samples_path()
        samples_df = load_training_samples(samples_path)
        temp_df = samples_df.head(1)

        boundaries_path_temp = config.get_boundaries_path()
        temp_dataset = LULCDataset(
satellite_path=config.get_satellite_path(),

```

```

    samples_df=temp_df,
    dem_path=config.get_dem_path(),
    nightlights_path=config.get_nightlights_path(), # Ночные
данные DMSP-OLS
    normalize=True,
    augment=False,
    patch_size=config.CNN_CONFIG['patch_size'],
    boundaries_path=boundaries_path_temp
    )
    sample_patch, _ = temp_dataset[0]
    input_channels = sample_patch.shape[0]

    cnn_config = config.CNN_CONFIG.copy()
    cnn_config['input_channels'] = input_channels
    # Удаляем параметры, которые не поддерживаются
обычной CNN
    cnn_config.pop('use_attention', None)

    model = CNN(**cnn_config)
    model.load_state_dict(torch.load(model_path))
    models.append(model)
    print(f" [OK] Модель {i+1} загружена:
{Path(model_path).name}")

    print()
else:
    print("Используется ОДИНОЧНАЯ модель")
    model_path = os.path.join(config.MODELS_DIR,
config.BEST_MODEL_NAME)
    if not Path(model_path).exists():
        print(f"[ОШИБКА] Модель не найдена:
{model_path}")
    print()
    print("Сначала обучите модель:")
    print(" python scripts/02_train_cnn_model.py")
    print("Или обучите ensemble:")
    print(" python scripts/03_train_ensemble.py")
    return

```

```

# Определяем размер входа
from utils import load_training_samples
samples_path = config.get_training_samples_path()
samples_df = load_training_samples(samples_path)
temp_df = samples_df.head(1)

temp_dataset = LULCDataset(
    satellite_path=config.get_satellite_path(),
    samples_df=temp_df,
    dem_path=config.get_dem_path(),
    nightlights_path=config.get_nightlights_path(), #
Ночные данные DMSP-OLS
    normalize=True,
    augment=False,
    patch_size=config.CNN_CONFIG['patch_size'], # Для
CNN используем patch_size
    boundaries_path=None
)
sample_patch, _ = temp_dataset[0]
input_channels = sample_patch.shape[0]

cnn_config = config.CNN_CONFIG.copy()
cnn_config['input_channels'] = input_channels
# Удаляем параметры, которые не поддерживаются
обычной CNN
cnn_config.pop('use_attention', None)

model = CNN(**cnn_config)
model.load_state_dict(torch.load(model_path))
models = [model]
print(f" [OK] Модель загружена:
{Path(model_path).name}")
print()

# Проверка наличия данных
try:
    satellite_path = config.get_satellite_path()

```

```

except ValueError as e:
    print(f"[ОШИБКА] {e}")
    return

dem_path = config.get_dem_path()
nightlights_path = config.get_nightlights_path()

print("Проверка файлов...")
if not Path(satellite_path).exists():
    print(f"[ОШИБКА] Спутниковые данные не найдены:
{satellite_path}")
    return
print(f" [OK] Спутниковые данные:
{Path(satellite_path).name}")

if dem_path and Path(dem_path).exists():
    print(f" [OK] DEM: {Path(dem_path).name}")
else:
    print(f" [INFO] DEM не указан или не найден")

if nightlights_path and Path(nightlights_path).exists():
    print(f" [OK] Ночные данные DMSP-OLS:
{Path(nightlights_path).name}")
else:
    print(f" [INFO] Ночные данные не указаны или не
найжены")
print()

# Путь для сохранения результата с указанием
архитектуры
satellite_name = Path(satellite_path).stem
model_arch = "CNN"
if use_ensemble:
    output_path = os.path.join(config.PREDICTIONS_DIR,
f'{satellite_name}_classified_{model_arch}_ensemble.tif')
else:
    output_path = os.path.join(config.PREDICTIONS_DIR,
f'{satellite_name}_classified_{model_arch}.tif')

```

```

# Загрузка границ Буланды (если указаны)
boundaries_path = config.get_boundaries_path()
if boundaries_path:
    print(f" [ОК] Используются границы Буланды:
{Path(boundaries_path).name}")
else:
    print(f" [INFO] Границы Буланды не указаны (можно
указать BOUNDARIES_FILE в config.py)")
print()

# Классификация
print("=" * 70)
print("НАЧАЛО КЛАССИФИКАЦИИ")
print("=" * 70)
print()

classify_image_ensemble(
    models=models,
    satellite_path=satellite_path,
    dem_path=dem_path,
    nightlights_path=nightlights_path, # Ночные данные
DMSP-OLS
    output_path=output_path,
    device=device,
    batch_size=1000,
    is_ensemble=use_ensemble,
    boundaries_path=boundaries_path # Маскирование по
границам Буланды
)

print("=" * 70)
print("КЛАССИФИКАЦИЯ ЗАВЕРШЕНА")
print("=" * 70)
print()
print("Все задачи выполнены в одном скрипте:")
print(f" ✓ Классификация выполнена")

```

```

print(f" ✓ GeoTIFF файл сохранен: {output_path}")
png_path = output_path.replace('.tif', '.png')
print(f" ✓ PNG визуализация создана: {png_path}")
print(f" ✓ Статистика отображена на изображении")
print(f" ✓ Точность оценена по классам")
print()
print("PNG изображение должно было открыться
автоматически.")
print("Вы также можете открыть GeoTIFF файл в QGIS
для дальнейшей работы.")

if __name__ == "__main__":
    main()

```

После того как вы вставили скрипт в файл .txt, сохраните изменения и измените формат на Python File (.py).

1: Откройте PowerShell и перейдите в папку CNN:

```

cd <ПУТЬ_К_ПАПКЕ_CNN>

```

2: Запустите скрипт классификации:

```

python scripts\04_classify_image.py

```

Шаг - 7.3. Что происходит во время классификации

Этап 1: Загрузка модели

КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ CNN МОДЕЛИ/ENSEMBLE

```

=====
Устройство: cuda

```

```

Используется ОДИНОЧНАЯ модель

```

```

[OK] Модель загружена: best_cnn_model.pth

```

```

Или для ensemble:

```

Используется ENSEMBLE моделей

Количество моделей: 3

[OK] Модель 1 загружена: ensemble_model_1.pth

[OK] Модель 2 загружена: ensemble_model_2.pth

[OK] Модель 3 загружена: ensemble_model_3.pth

...

Этап 2: Загрузка данных

...

Загрузка спутниковых данных...

[OK] Спутниковые данные: (6, 3471, 6921)

Каналов: 6

Размер: 3471 x 6921

Фильтрация выборки по границам Буланды...

[OK] Shapefile загружен: 1 объектов

CRS: EPSG:4326

Вычисление вегетационных индексов...

[OK] Вычислено индексов: 5

Загрузка DEM данных...

[OK] DEM: (3457, 6921)

Загрузка ночных данных DMSP-OLS...

[OK] Ночные данные: (3457, 6921)

...

Этап 3: Классификация

...

[INFO] Классификация будет выполняться только внутри границ Буланды

Пиксели вне границ будут помечены как фон (класс 0)

Обработка пикселей батчами по 1000...

Классификация:

100%

9220/9220 [32:15<00:00, 4.75it/s]

...

Этап 4: Сохранение результатов

...

Сохранение результата в

...Bulandy_Composite_2000_Full6Bands_classified_CNN.tif...

[OK] Результат сохранен:
...Bulandy_Composite_2000_Full6Bands_classified_CNN.tif
Статистика классификации (только область Буланды):

Класс 1 (Пашня): 3,998,654 пикселей (43.37%)
Класс 2 (Пастбища): 2,292,494 пикселей (24.87%)
Класс 3 (Леса): 1,476,168 пикселей (16.01%)
Класс 4 (Вода): 680,939 пикселей (7.39%)
Класс 5 (Застроенные территории): 770,770 пикселей
(8.36%)
...

Этап 5: Создание визуализации
...

Создание PNG визуализации...

[OK] Изображение открыто автоматически

[OK] Визуализация сохранена:

...Bulandy_Composite_2000_Full6Bands_classified_CNN.png
...

Шаг - 7.4. Результаты классификации

После завершения классификации в папке

`results/predictions/` будут сохранены:

- `\*\_classified\_CNN.tif` - GeoTIFF файл с классификацией
- `\*\_classified\_CNN.png` - PNG визуализация с картой и

статистикой

Шаг - 7.5. Работа с результатами

Открытие GeoTIFF в QGIS:

1. Откройте QGIS
2. Добавьте слой → Растровый слой
3. Выберите файл `\*\_classified\_CNN.tif`
4. Настройте цветовую схему для классов

Открытие PNG:

- PNG файл открывается автоматически после
классификации

- Содержит карту классификации и статистику

Шаг - 8. ИНТЕРПРЕТАЦИЯ РЕЗУЛЬТАТОВ

Шаг - 8.1. Статистика классификации

Распределение классов:

- Показывает процентное соотношение каждого класса в области Буланды
- Рассчитывается только для области внутри границ (класс 0 - фон исключается)
- Точность по классам:
 - Показывает, насколько хорошо классифицируется каждый класс
 - Рассчитывается на основе обучающей выборки

Шаг - 8.2. Визуальная оценка

Проверьте:

- Соответствие границам области
- Логичность распределения классов
- Отсутствие артефактов и шума
- Корректность идентификации застроенных территорий

Шаг - 9. РЕШЕНИЕ ПРОБЛЕМ

Проблема: "Модель не найдена"

Ошибка:

```
'''
```

```
[ОШИБКА] Модель не найдена: ...best_cnn_model.pth
```

```
'''
```

Решение:

1. Убедитесь, что обучение завершилось успешно
2. Проверьте наличие файла:

```
```powershell
dir results\models\best_cnn_model.pth
```
```

3. Если файла нет, запустите обучение:

```
```powershell
python scripts\02_train_cnn_model.py
```
```

Проблема: "Недостаточно памяти"

Ошибка:

```
'''
```

```
RuntimeError: CUDA out of memory
```

```
'''
```

Решение:

1. Уменьшите `BATCH\_SIZE` в `config.py`:

```
```python
```

```
BATCH_SIZE = 32 # Было 64
'''
```

2. Уменьшите размеры сети:

```
'''python
'hidden_sizes': [256, 128, 64] # Было [512, 256, 128, 64]
'''
```

3. Отключите контекстные признаки (не рекомендуется):

```
'''python
USE_CONTEXT_FEATURES = False
'''
```

### **Проблема: "Sharfile не найден"**

Ошибка:

```
'''
```

[WARNING] Границы Буланды не указаны или не найдены

```
'''
```

Решение:

1. Проверьте путь в `config.py`:

```
'''python
BOUNDARIES_FILE = os.path.join(DATA_DIR,
'boundaries', 'bulandy_boundaries_from_raster.shp')
'''
```

2. Убедитесь, что файл существует

3. Используйте абсолютный путь

### **Проблема: "Низкая точность (< 60%)"**

Решение:

1. Увеличьте `LEARNING\_RATE` до `0.001`

2. Увеличьте размеры сети: `[1024, 512, 256, 128, 64]`

3. Уменьшите `dropout` до `0.1`

4. Увеличьте `NUM\_EPOCHS` до `300`

5. Используйте расширенную выборку (2500+ точек на класс)

6. Проверьте качество обучающей выборки

### **Проблема: "Переобучение (train accuracy >> val accuracy)"**

Решение:

1. Увеличьте `dropout` до `0.3` или `0.4`

2. Увеличьте `WEIGHT\_DECAY` до `2e-4`

3. Уменьшите размеры сети: `[256, 128, 64]`

4. Уменьшите `LEARNING\_RATE` до `0.0003`
5. Уменьшите `EARLY\_STOPPING\_PATIENCE` до `20`

## Шаг - 10. ПОЛНАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ

### РАБОТЫ

#### 1: Подготовка данных

1. Поместите все исходные данные в соответствующие папки

2. Проверьте структуру папок
3. Настройте `config.py` (укажите пути к данным)

#### 2: Визуализация границ (рекомендуется)

Создайте файл формата .txt в папке scripts и назовите его 00\_visualize\_boundaries. Далее откройте файл через «Блокнот» и вставьте скрипт:

```
"""Визуализация границ Буланды и проверка обучающей
выборки"""
```

```
import sys
sys.stdout.reconfigure(encoding='utf-8')
```

```
import os
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
import rasterio
from rasterio.plot import show
```

```
Добавляем родительскую директорию в путь для импорта
parent_dir =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
sys.path.insert(0, parent_dir)
import config
```

```
try:
 import geopandas as gpd
 from shapely.geometry import Point
 HAS_GEOPANDAS = True
except ImportError:
 HAS_GEOPANDAS = False
```

```

 print("[ОШИБКА] geopandas не установлен. Установите:
pip install geopandas shapely")
 sys.exit(1)

```

```

def visualize_boundaries_and_samples():
 """Визуализирует границы Буланды и обучающую
выборку"""
 print("=" * 70)
 print("ВИЗУАЛИЗАЦИЯ ГРАНИЦ БУЛАНДЫ И
ПРОВЕРКА ВЫБОРКИ")
 print("=" * 70)
 print()

```

```

 # Загрузка границ
 boundaries_path = config.get_boundaries_path()
 if not boundaries_path or not Path(boundaries_path).exists():
 print("[ОШИБКА] Shapefile границ не найден!")
 print(f" Путь: {boundaries_path}")
 print()
 print("Укажите путь к shapefile в config.py:")
 print(" BOUNDARIES_FILE = r'путь\\к\\файлу.shp'")
 return

 print(f"Загрузка границ из: {Path(boundaries_path).name}")
 boundaries_gdf = gpd.read_file(boundaries_path)
 print(f" [OK] Загружено {len(boundaries_gdf)}
полигонов")
 print(f" CRS: {boundaries_gdf.crs}")
 print()

```

```

 # Загрузка спутниковых данных для фона
 satellite_path = config.get_satellite_path()
 if not Path(satellite_path).exists():
 print("[ОШИБКА] Спутниковые данные не найдены!")
 return

```

```

 print(f"Загрузка спутниковых данных:
{Path(satellite_path).name}")

```

```

with rasterio.open(satellite_path) as src:
 satellite_crs = src.crs
 transform = src.transform
 # Читаем RGB каналы для визуализации
 if src.count >= 3:
 red = src.read(3) # Red
 green = src.read(2) # Green
 blue = src.read(1) # Blue
 rgb = np.dstack([red, green, blue])
 # Нормализация для визуализации
 rgb = np.clip(rgb / rgb.max() * 255, 0,
255).astype(np.uint8)
 else:
 rgb = src.read(1) # Одноканальный

 # Перепроецируем границы в CRS спутниковых данных
 if boundaries_gdf.crs != satellite_crs:
 print(f'Перепроецирование границ из
{boundaries_gdf.crs} в {satellite_crs}...')
 boundaries_gdf = boundaries_gdf.to_crs(satellite_crs)

 print()

 # Загрузка обучающей выборки
 from utils import load_training_samples
 samples_path = config.get_training_samples_path()
 print(f'Загрузка обучающей выборки:
{Path(samples_path).name}')
 samples_df = load_training_samples(samples_path)
 print(f' [OK] Загружено {len(samples_df)} точек')
 print()

 # Проверка точек внутри границ
 print("Проверка точек внутри границ Буланды...")
 if len(boundaries_gdf) > 1:
 union_geom = boundaries_gdf.geometry.unary_union
 else:
 union_geom = boundaries_gdf.geometry.iloc[0]

```

```

points_inside = []
points_outside = []

for idx, row in samples_df.iterrows():
 point = Point(row['x'], row['y'])
 if union_geom.contains(point):
 points_inside.append((row['x'], row['y'], row['class']))
 else:
 points_outside.append((row['x'], row['y'], row['class']))

print(f" Точки внутри границ: {len(points_inside)}")
print(f" Точки вне границ: {len(points_outside)}")
print()

if len(points_outside) > 0:
 print("[ВНИМАНИЕ] Найдены точки вне границ
Буланды!")
 print(" Эти точки будут исключены при обучении.")
 print()

Создание визуализации
print("Создание визуализации...")
fig, ax = plt.subplots(figsize=(16, 12), dpi=150)

Отображаем спутниковые данные
if len(rgb.shape) == 3:
 show(rgb, ax=ax, transform=transform, cmap='gray' if
rgb.shape[2] == 1 else None)
else:
 show(rgb, ax=ax, transform=transform, cmap='gray')

Отображаем границы
boundaries_gdf.plot(ax=ax, edgecolor='red', facecolor='none',
linewidth=2,
label='Границы Буланды', linestyle='--')

Отображаем точки
if points_inside:

```

```

 inside_x = [p[0] for p in points_inside]
 inside_y = [p[1] for p in points_inside]
 inside_classes = [p[2] for p in points_inside]
 scatter_inside = ax.scatter(inside_x, inside_y,
c=inside_classes,
 cmap='tab10', s=50, alpha=0.7,
 edgecolors='white', linewidths=1,
 label=f'Точки внутри ({len(points_inside)})',
 zorder=5)

 if points_outside:
 outside_x = [p[0] for p in points_outside]
 outside_y = [p[1] for p in points_outside]
 ax.scatter(outside_x, outside_y, c='red', s=100, marker='x',
 linewidths=3, label=f'Точки вне границ
({len(points_outside)})',
 zorder=6)

 ax.set_title('Границы Буланды и Обучающая Выборка',
fontsize=16, fontweight='bold')
 ax.legend(loc='upper right', fontsize=10)
 ax.set_xlabel('Долгота', fontsize=12)
 ax.set_ylabel('Широта', fontsize=12)

Сохранение
output_path = os.path.join(config.RESULTS_DIR,
'boundaries_visualization.png')
os.makedirs(config.RESULTS_DIR, exist_ok=True)
plt.savefig(output_path, dpi=150, bbox_inches='tight')
print(f" [OK] Визуализация сохранена: {output_path}")

Автоматически открываем
try:
 if os.name == 'nt':
 os.startfile(output_path)
 print(" [OK] Изображение открыто автоматически")
except:
 pass

```

```

plt.close()

print()
print("=" * 70)
print("РЕЗУЛЬТАТЫ ПРОВЕРКИ")
print("=" * 70)
print(f"Всего точек в выборке: {len(samples_df)}")
print(f"Точек внутри границ: {len(points_inside)}")
print(f"({len(points_inside)/len(samples_df)*100:.1f}%)")
print(f"Точек вне границ: {len(points_outside)}")
print(f"({len(points_outside)/len(samples_df)*100:.1f}%)")
print()

if len(points_outside) == 0:
 print("[ОК] Все точки находятся внутри границ
Буланды!")
else:
 print("[ВНИМАНИЕ] Некоторые точки находятся вне
границ и будут исключены.")
 print(" Рекомендуется обновить обучающую выборку,
чтобы все точки были внутри.")
 print()
 if __name__ == "__main__":
 visualize_boundaries_and_samples()

```

После того как вы вставили скрипт в файл .txt, сохраните изменения и измените формат на Python File (.py)

```

cd <ПУТЬ_К_ПАПКЕ_CNN>
python scripts\00_visualize_boundaries.py

```

Результат: PNG изображение с границами и обучающими точками

### **3: Расширение выборки (рекомендуется)**

```

python scripts\00_expand_training_samples.py

```

Результат: `training\_samples\_expanded.csv` (2500 точек на класс)

Обновите config.py:

```
python TRAINING_SAMPLES_FILE =
'training_samples_expanded.csv'
```

#### **4: Проверка данных**

...

```
python scripts\01_check_training_samples.py
```

...

Результат: Отчет о проверке данных

#### **5: Настройка параметров**

1. Откройте `config.py`
2. Настройте параметры обучения и модели (см. раздел 4)
3. Сохраните файл

#### **6: Обучение одиночной модели**

...

```
python scripts\02_train_cnn_model.py
```

...

Результат:

- `results/models/best\_cnn\_model.pth` - лучшая модель
- `results/models/cnn\_final\_model.pth` - финальная модель
- Отчет о точности на тестовых данных

Время обучения: 30-60 минут (зависит от размера выборки и GPU)

#### **7: Обучение Ensemble (опционально, но рекомендуется)**

...

```
python scripts\03_train_ensemble.py
```

...

Результат:

- `results/models/ensemble\_model\_\*.pth` - модели ensemble
- `results/models/ensemble\_info.json` - информация об ensemble

ensemble

- Отчет о точности ensemble

Время обучения: 1.5-3 часа (зависит от количества моделей)

#### **8: Классификация изображения**

...

```
python scripts\04_classify_image.py
```

'''

Результат:

- `results/predictions/\*\_classified\_CNN.tif` - GeoTIFF с  
классификацией

- `results/predictions/\*\_classified\_CNN.png` - PNG  
визуализация

- Статистика классификации

- PNG изображение открывается автоматически

Время классификации: 30-60 минут (зависит от размера  
изображения)

## Шаг - 11. РЕКОМЕНДУЕМЫЕ ПАРАМЕТРЫ ДЛЯ РАЗНЫХ СЦЕНАРИЕВ

**Сценарий 1: Быстрое обучение (низкая точность  
приемлема)**

```
'''python
LEARNING_RATE = 0.001
BATCH_SIZE = 128
NUM_EPOCHS = 100
CNN_CONFIG = {
 'hidden_sizes': [256, 128, 64],
 'dropout': 0.2,
}
'''
```

**### Сценарий 2: Баланс скорости и точности  
(рекомендуется)**

```
'''python
LEARNING_RATE = 0.0005
BATCH_SIZE = 64
NUM_EPOCHS = 200
CNN_CONFIG = {
 'hidden_sizes': [512, 256, 128, 64],
 'dropout': 0.2,
}
'''
```

**### Сценарий 3: Максимальная точность**

```
'''python
LEARNING_RATE = 0.0005
BATCH_SIZE = 64
```

```

NUM_EPOCHS = 300
CNN_CONFIG = {
 'hidden_sizes': [1024, 512, 256, 128, 64],
 'dropout': 0.25,
}
CONTEXT_WINDOW_SIZE = 5
Использовать расширенную выборку (2500+ точек на
класс)
Обучить ensemble из 5 моделей
'''

```

## Шаг - 12. ПРИМЕРЫ КОМАНД ДЛЯ ТЕРМИНАЛА

**ВАЖНО:** Перед выполнением команд замените  
`<ПУТЬ\_К\_ПАПКЕ\_CNN>` на реальный путь к папке CNN на  
вашем компьютере.

Пример:

- Если папка находится на рабочем столе: `cd  
C:\Users\ВашеИмя\Desktop\DL method\CNN`
- Если папка находится в другом месте: `cd D:\Projects\DL  
method\CNN`
- Или используйте относительный путь от текущей  
директории

Как узнать путь к папке:

1. Откройте проводник Windows
2. Перейдите в папку `CNN`
3. Кликните в адресной строке - путь будет выделен
4. Скопируйте путь (Ctrl+C)
5. Используйте этот путь в команде `cd`

'''

```

Переход в папку CNN
cd <ПУТЬ_К_ПАПКЕ_CNN>

```

```

1: Визуализация границ
python scripts\00_visualize_boundaries.py

```

```

2: Расширение выборки
python scripts\00_expand_training_samples.py

```

```
3: Проверка данных
python scripts\01_check_training_samples.py
```

```
4: Обучение модели
python scripts\02_train_cnn_model.py
```

```
5: Обучение ensemble (опционально)
python scripts\03_train_ensemble.py
```

```
6: Классификация
python scripts\04_classify_image.py
'''
```

```
Только обучение и классификация:
'''
```

```
cd <ПУТЬ_К_ПАПКЕ_CNN>
python scripts\02_train_cnn_model.py
python scripts\04_classify_image.py
'''
```

### **3.2 Преимущества, ограничения, перспективы глубокого обучения**

Преимущества. Алгоритмы глубокого обучения обладают рядом преимуществ для классификации LULC: автоматическое извлечение признаков, что устраняет необходимость в ручном проектировании признаков; способность обрабатывать и интегрировать разнородные источники данных (мультиспектральные, радарные, LiDAR); высокая точность классификации, особенно в сложных городских ландшафтах и возможность обучения сквозных моделей, объединяющих предобработку данных и классификацию.

Ограничения. Несмотря на значительные преимущества, использование глубокого обучения для LULC классификации сопряжено с некоторыми ограничениями: потребность в больших объемах размеченных данных для обучения; высокие вычислительные требования, особенно для обучения на больших наборах данных; сложность интерпретации результатов работы "черного

ящика" нейронных сетей и риск переобучения на небольших или несбалансированных наборах данных.

Перспективы. Развитие методов глубокого обучения для классификации LULC связано с: разработкой архитектур, способных эффективно обрабатывать временные ряды спутниковых снимков; применением методов переноса обучения для адаптации моделей к новым регионам; интеграцией экспертных знаний и глубокого обучения для повышения интерпретируемости моделей; разработкой методов обучения с малым количеством размеченных данных (few-shot learning). Алгоритмы глубокого обучения для классификации LULC могут быть эффективно использованы для: высокоточного картирования сельскохозяйственных земель и типов культур; мониторинга изменений землепользования в режиме реального времени; оценки состояния посевов и прогнозирования урожайности; выявления аномалий и раннего предупреждения о потенциальных проблемах на сельскохозяйственных угодьях. Применение алгоритмов глубокого обучения в сочетании с облачными платформами обработки данных, такими как Google Earth Engine, открывает новые возможности для точного и эффективного картирования LULC на больших территориях, что особенно актуально для масштабов Республики Казахстан. Дальнейшее развитие алгоритмов классификации LULC связано с: использованием глубокого обучения и сверточных нейронных сетей; интеграцией разнородных источников данных (мультиспектральные, радарные, LiDAR) и разработкой методов интерпретации результатов работы сложных моделей. Применение алгоритмов классификации в сочетании с современными платформами обработки данных открывает новые возможности для точного и эффективного картирования LULC на больших территориях.

## Глава 4. СРАВНЕНИЕ МОДЕЛЕЙ ДЛЯ КЛАССИФИКАЦИИ LULC

Maximum Likelihood Classification относит пиксель к тому классу, вероятность принадлежности к которому максимальна, оценивая для каждого класса вектор средних и ковариационную матрицу и предполагая многомерное нормальное (гауссово) распределение признаков. Преимущества: теоретически близок к байесовскому «оптимуму» при выполнении допущений; учитывает форму и ориентировку облака точек, поэтому хорошо разделяет классы с разной дисперсией/корреляцией каналов; даёт вероятностный вывод, позволяет учитывать априорные вероятности и «стоимость ошибок»; обычно превосходит простые геометрические правила при хорошо обученных сигнатурах. Недостатки: чувствителен к нарушению нормальности и мульти-модальности внутри класса (фенология, увлажнение, тени); требует много репрезентативных обучающих пикселей (для устойчивой оценки ковариации - существенно больше числа признаков), иначе возможны вырожденные/неустойчивые матрицы; требует нормализации, отбора признаков и преобразования минимальной доли шума; вычислительно тяжелее MD/SAM; на «смешанных» пикселях и при сильной гетерогенности ландшафта даёт смазывание границ и рост путаницы между близкими классами.

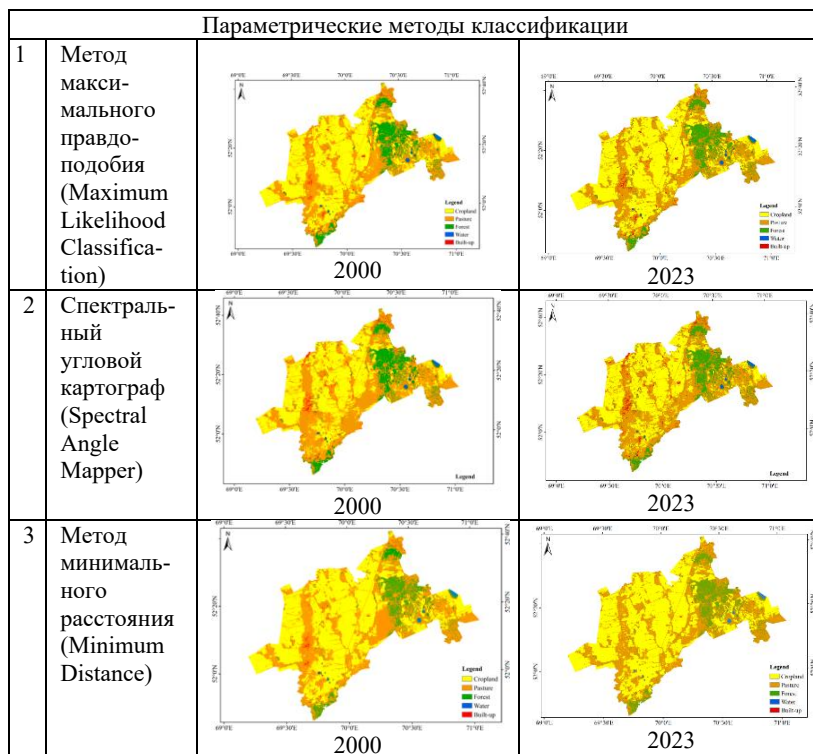
Spectral Angle Mapper. SAM сравнивает угол между спектральным вектором пикселя и эталонным спектром класса в N-мерном пространстве; чем меньше угол, тем выше сходство формы спектральной подписи. Преимущества: инвариантность к масштабу яркости - поэтому особенно полезен для гиперспектральных данных и библиотек эндмемберов (эталонные векторы, с которыми метод сравнивает спектр каждого пикселя, измеряя угол между ними); простая интерпретация «схожести по форме» спектра; умеренные требования к объёму обучающих данных; стабилен при добавлении «информативных» полос. Недостатки: игнорирует модуль яркости - различия амплитуд (альbedo) не используются, что при низком SNR/атмосферных остатках ведёт к путанице; требует хорошо выделенных «чистых» эндмемберов, чувствителен к внутриклассовой вариабельности

(влажность, фенология, почвенный фон); для широкополосных мультиспектральных данных (например Landsat) может уступать MLC/ML-подходам, если не настроен порог угла и не расширен признаками (NDVI/NDWI, текстуры); не выдаёт вероятности, важен выбор порога «неопознанных» пикселей; хуже работает на смешанных пикселях и при сильной спектральной близости материалов.

Minimum Distance. MD присваивает пиксель классу с минимальным евклидовым расстоянием до среднего спектра класса; фактически это линейное правило ближайшего среднего и эквивалентно MLC при равных сферических ковариациях. Преимущества: простота настройки и очень высокая скорость; работает при малом объеме обучающих данных, не требует оценки ковариаций; прозрачен и воспроизводим, удобен как базовый ориентир; устойчив к небольшим шумам при правильной нормализации признаков. Недостатки: игнорирует различия в дисперсиях и корреляциях каналов; без стандартизации чувствителен к масштабам признаков; решение задаётся плоскими границами и плохо справляется с вытянутыми/перекрывающимися облаками; склонен к «сдвигу» из-за выбросов, меняющих центр; не выдаёт вероятностей/уверенности; на сложных сценах и при спектральной близости классов обычно менее точен, чем MLC/SAM/ML-модели.

На рисунке 4.1 представлен визуальное сравнение трех вышеприведенных параметрических алгоритмов классификации LULC Буландинского района за 2000 и 2023 годы.

Classification and Regression Trees. Классическое дерево решений, основанное на построении иерархической структуры в виде дерева, где каждый узел представляет собой решение, основанное на одном из признаков входных данных, а листья дерева - конечные классы, к которым относится тот или иной пиксель изображения. Для LULC это даёт мгновенно интерпретируемые правила (например, “ $NDVI > 0.42$  и  $NDWI < 0.05 \Rightarrow$  Лес”), позволяя понять вклад индексов (NDVI, NDWI и т.д.), спектральных каналов и рельефных/текстурных признаков.



**Рисунок 4.1** - Визуальное сравнении параметрических алгоритмов классификации

Плюсы: высокая прозрачность и объяснимость; минимальная подготовка (масштабирование обычно не требуется); устойчивость к нелинейностям и взаимодействиям признаков; есть механизмы учёта стоимости ошибок и дисбаланса классов (class weights). Минусы: высокая дисперсия (нестабильность к небольшим изменениям обучающей выборки), склонность к переобучению без грамотной обрезки; чувствительность к шумным меткам; часто уступает ансамблевым, ядовым методам по точности на сложных ландшафтах. Практически CART - отличный базовый ориентир и инструмент диагностики признаков; для производственных карт LULC его часто усиливают пост-обработкой (фильтры

моды/морфология/сегментация) или заменяют ансамблевыми моделями обучения.

**Random Forest.** Ансамбль из множества деревьев, обученных на основе бэггинга (bagging) – техники ансамблевого обучения, при которой создается множество деревьев решений, каждое из которых обучается на случайной подвыборке обучающих данных. Плюсы: высокая точность и обобщающая способность на разнотипных данных (мультиспектр + индексы + текстуры + DEM); робастность к шуму и выбросам; почти не требует масштабирования; встроенная оценка точности без отложенной; важности признаков помогают отбору каналов/индексов. Минусы: меньшая интерпретируемость (правил много, и они разнородны); потенциальный перекося в сторону доминирующих классов без балансировки; вероятности часто плохо калиброваны; на очень высоких разрешениях и с большим числом деревьев возрастает время и память; границы решений кусочно-постоянны, что может давать «пиксельный шум» без контекстной фильтрации/сегментации. В целом RF – надёжный выбор по умолчанию для LULC, особенно при множестве признаков и умеренно большом датасете.

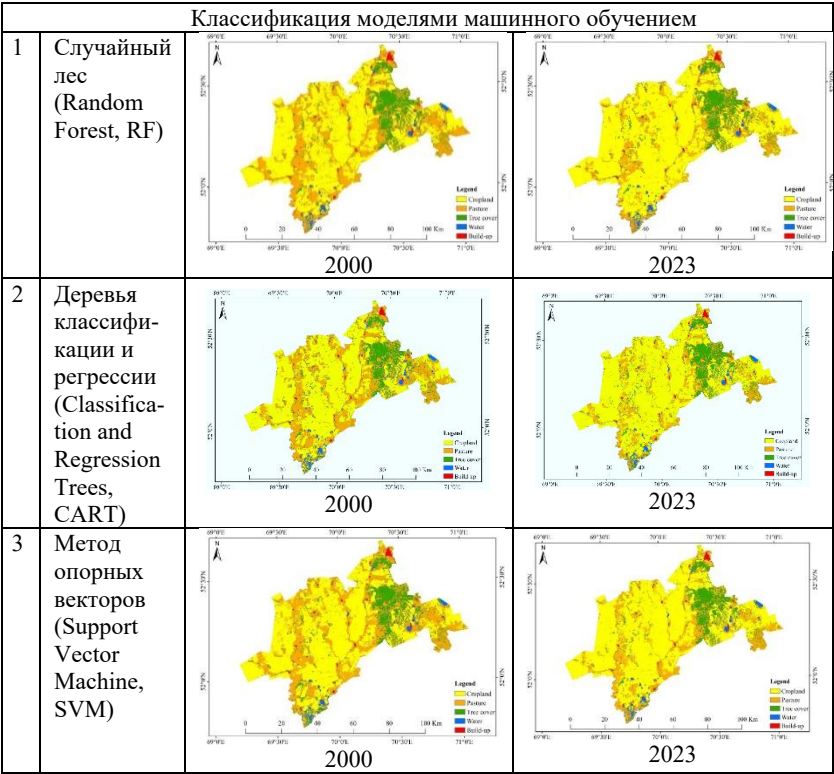
**Support Vector Machine. SVM** работает путем нахождения оптимальной гиперплоскости, которая максимально разделяет классы, обеспечивая при этом наибольший зазор (margin) между точками, принадлежащими разным классам. Такие точки называются опорными векторами и играют ключевую роль в построении модели. В LULC особенно силён при невысоком числе обучающих образцов и высокой размерности признаков. Плюсы: высокая точность на сложных, частично перекрывающихся классах; гибкость за счёт выбора ядра и регуляризации ( $C$ ) и масштаба ядра ( $\gamma$ ); хорошо работает в сценариях при большом количестве признаков и не большого количества эталонов. Минусы: требует обязательного масштабирования признаков; чувствителен к настройке  $C$  и  $\gamma$  (нужен тщательный поиск по сетке/байесовская оптимизация); обучение и применение уже обученной модели может быть тяжелым на больших выборках; интерпретируемость низкая; чувствителен к шумным меткам и дисбалансу, решается данная проблема методами `class_weight` (задать веса классов в функции потерь, чтобы модель не «игнорировала» малочисленные классы)/ `robust`

С (подобрать регуляризацию С так, чтобы модель была менее чувствительна к выбросам/шуму).

На рисунке 4.2 представлено визуальное сравнение трех вышеприведенных алгоритмов классификации LULC Буландинского района на основе машинного обучения за 2000 и 2023 годы.

Artificial Neural Network представляет собой обобщённую архитектуру искусственных нейронных сетей, способную моделировать сложные нелинейные зависимости между спектральными, текстурными и дополнительными признаками дистанционного зондирования. В задачах классификации LULC ANN применяются для обработки многомерных входных данных (спектральные каналы, индексы NDVI/NDWI/EVI, DEM-производные, текстуры), обеспечивая более гибкие границы решений по сравнению с классическими параметрическими алгоритмами. Преимущества ANN заключаются в способности аппроксимировать сложные нелинейные зависимости, устойчивости к коррелированным признакам и возможности учитывать взаимодействия между ними. Недостатки включают необходимость большого объёма обучающих данных, чувствительность к настройке архитектуры и гиперпараметров, а также меньшую интерпретируемость результатов по сравнению с деревьями решений.

Multi-Layer Perceptron является частным случаем ANN и представляет собой полносвязную нейронную сеть с несколькими скрытыми слоями. В классификации LULC MLP особенно эффективен при использовании заранее вычисленных признаков (спектральные индексы, статистики, текстуры), поскольку не учитывает пространственный контекст напрямую, а работает с вектором признаков каждого пикселя. Преимущества MLP включают высокую точность при наличии достаточного числа обучающих примеров, способность выявлять нелинейные зависимости и относительно простую реализацию. Однако MLP уступает ансамблевым методам (RF, SVM) при небольших обучающих выборках и может переобучаться на гетерогенных сценах без регуляризации. По аналогии с результатами, представленными в документе, MLP обычно демонстрирует точность, сопоставимую с SVM или немного ниже RF, поскольку, как и ядровые методы, способен формировать сложные нелинейные границы классов, но менее устойчив к шуму и дисбалансу классов.



**Рисунок 4.2** - Визуальное сравнении классификаций алгоритмов машинного обучения

Convolutional Neural Network является наиболее продвинутым типом ANN для анализа изображений, поскольку использует свёрточные фильтры для автоматического извлечения пространственных, текстурных и контекстных признаков. В отличие от MLP и классических ANN, CNN учитывает пространственную структуру данных, что особенно важно для LULC, где соседние пиксели часто принадлежат одному классу. Преимущества CNN включают высокую точность на сложных и гетерогенных сценах, способность выявлять пространственные паттерны (например, урбанизированные структуры, лесные массивы, сельскохозяйственные поля) и снижение «пиксельного шума» классификации. Недостатки - высокая вычислительная сложность, потребность в

178

больших обучающих наборах и сложность настройки архитектуры. В большинстве исследований CNN превосходит традиционные методы машинного обучения, включая RF и SVM, особенно при использовании данных высокого пространственного разрешения.

Визуальное сравнение трех вышеприведенных алгоритмов глубокого обучения за 2000 и 2023 годы приведено на рисунке 4.3.

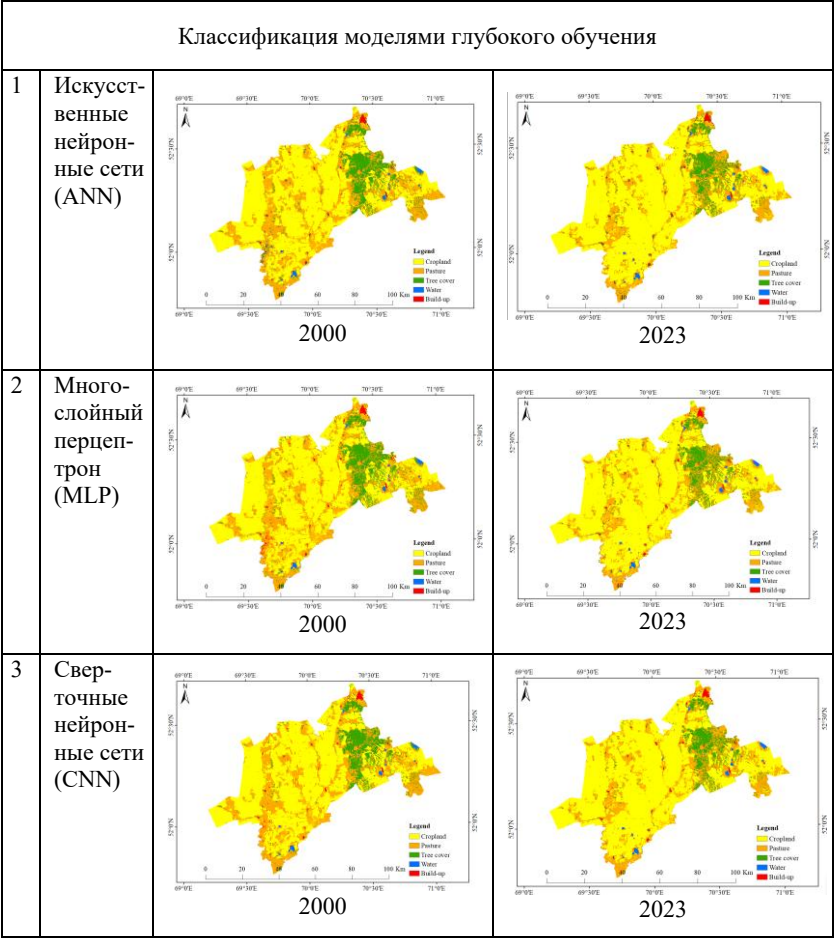


Рисунок 4.3 - Визуальное сравнение классификаций алгоритмов глубокого обучения

В целом визуальное сравнение отдельных алгоритмов показали чёткое преимущество методов машинного обучения над классическими параметрическими алгоритмами при классификации LULC на рассматриваемых данных.

Учитывая, что визуальное сравнение относительно малоинформативны, в таблице 4.1 приведены точности классификации LULC каждым алгоритмом. Как видно из таблицы 4.1 наибольшую точность демонстрирует RF: overall accuracy 0,94, producer's accuracy 0,92, user's accuracy 0,93 и  $\kappa=0,91$ . Очень близко к нему идёт SVM (0,93/0,91/0,92;  $\kappa=0,90$ ), а CART немного уступает, но остаётся сильным базовым вариантом (0,91/0,89/0,90;  $\kappa=0,88$ ).

*Таблица 4.1*

**Точность классификации параметрическими алгоритмами, методами машинного и глубокого обучения**

Алгоритмы классификации	Overall Accuracy	Producer's Accuracy	User's Accuracy	Kappa Statistic, $\kappa$
MLC	$0.88 \pm 0.01$	$0.85 \pm 0.01$	$0.86 \pm 0.01$	$0.84 \pm 0.01$
SAM	$0.82 \pm 0.01$	$0.80 \pm 0.01$	$0.79 \pm 0.01$	$0.78 \pm 0.01$
MD	$0.79 \pm 0.01$	$0.76 \pm 0.01$	$0.75 \pm 0.01$	$0.72 \pm 0.01$
RF	$0.94 \pm 0.02$	$0.92 \pm 0.01$	$0.93 \pm 0.01$	$0.91 \pm 0.01$
CART	$0.91 \pm 0.01$	$0.89 \pm 0.01$	$0.90 \pm 0.01$	$0.88 \pm 0.01$
SVM	$0.93 \pm 0.03$	$0.91 \pm 0.01$	$0.92 \pm 0.01$	$0.90 \pm 0.01$
ANN	$0.87 \pm 0.01$	$0.85 \pm 0.01$	$0.88 \pm 0.01$	$0.82 \pm 0.01$
MLP	$0.89 \pm 0.01$	$0.88 \pm 0.01$	$0.90 \pm 0.01$	$0.86 \pm 0.01$
CNN	$0.91 \pm 0.01$	$0.91 \pm 0.01$	$0.90 \pm 0.01$	$0.89 \pm 0.01$

Высокие значения  $k$  для всех трёх методов (0,88–0,91) указывают на существенное превосходство над случайным совпадением и устойчивость результатов на гетерогенных сценах. Такой разрыв объясним: ансамбли и ядровые методы гибко аппроксимируют сложные, нелинейные границы классов, учитывают взаимодействия признаков (спектральные каналы, индексы NDVI/NDWI/EVI, текстуры, рельеф), менее чувствительны к отклонениям от нормальности распределений и шуму, а RF вдобавок обеспечивает устойчивость к переобучению за счёт бэггинга и случайного выбора признаков.

Среди параметрических алгоритмов лидирует MLC с overall 0,88, producer's 0,85, user's 0,86 и  $k=0,84$ . Это подтверждает его репутацию «сильного классика», особенно когда классы в признаковом пространстве близки к многомерному нормальному распределению и ковариации корректно оценены. SAM показывает средние результаты (0,82/0,80/0,79;  $k=0,78$ ): метод устойчив к вариациям освещённости, поскольку использует углы между спектральными векторами, но хуже различает классы с похожей формой спектра (например, пастбища и пашня) и не учитывает дисперсию/ковариацию. Наименее точен Minimum Distance (0,79/0,76/0,75;  $k=0,72$ ), что ожидаемо: он опирается только на евклидово расстояние до центроидов и игнорирует различия в разбросе признаков - из-за этого путает классы при наложении облаков точек и в присутствии коррелированных каналов.

Если сопоставить группы алгоритмов машинного обучения и параметрических алгоритмов, выигрыш очевиден: при переходе от MLC к RF overall растёт с 0,88 до 0,94 ( $\Delta=+0,06$ ),  $k$  - с 0,84 до 0,91 ( $\Delta=+0,07$ ); SVM даёт сходный прирост (0,93;  $k=0,90$ ), а даже простой CART обгоняет MLC по всем метрикам (0,91;  $k=0,88$ ). При этом у RF и SVM выше и producer's, и user's точности (0,91–0,93 против 0,85–0,86 у MLC), то есть они одновременно лучше восстанавливают истинные пиксели классов и дают более «чистые» пользовательские классы на карте. Практически это означает меньше пропусков лесов и водных объектов, и меньше «примесей»

в масках пашни и застройки. Тем не менее, параметрические методы сохраняют ценность: MLC - когда обучающая выборка мала, а распределения классов близки к гауссовым; SAM - при наличии качественных эталонных спектров и сильных перепадах освещённости; MD - как простой и быстрый эталон для сравнения по классификации. Является минимально приемлемой отправной точкой, на фоне которой вы оцениваете, есть ли смысл в более сложных моделях классификации. Для производственных карт LULC в условиях сложного ландшафта и смешанных пикселей рационально отдавать приоритет RF/SVM (с балансировкой классов, калибровкой вероятностей и пост-фильтрацией), а параметрические алгоритмы использовать как интерпретируемый бенчмарк и для валидации устойчивости результатов.

Дополнительно в исследовании были рассмотрены нейросетевые модели ANN, MLP и CNN. Классическая искусственная нейронная сеть ANN показала умеренно высокую точность (0,87/0,85/0,88;  $\kappa=0,82$ ), уступая ансамблевым и ядровым методам, что связано с её чувствительностью к объёму обучающих данных и сложности настройки архитектуры. Модель MLP, являющаяся многослойной полносвязной нейронной сетью, продемонстрировала более высокие результаты (0,89/0,88/0,90;  $\kappa=0,86$ ), приблизившись к CART и превзойдя параметрические алгоритмы. Это объясняется способностью MLP выявлять сложные нелинейные зависимости между спектральными признаками и индексами, однако отсутствие учёта пространственного контекста ограничивает её эффективность на гетерогенных сценах.

Наиболее высокую точность среди нейросетевых методов показала сверточная нейронная сеть CNN (0,91/0,91/0,90;  $\kappa=0,89$ ), сопоставимую с CART и близкую к SVM. Преимущество CNN обусловлено способностью автоматически извлекать пространственные и текстурные признаки из изображений, что особенно важно при классификации LULC, где соседние пиксели часто принадлежат одному классу. Благодаря учёту пространственного контекста CNN лучше справляется со смешанными пикселями, сложной структурой ландшафта и

неоднородностью сцен, уменьшая «пиксельный шум» и повышая целостность выделенных объектов.

Среди параметрических алгоритмов лидирует MLC с overall 0,88, producer's 0,85, user's 0,86 и  $\kappa=0,84$ . Это подтверждает его репутацию «сильного классика», особенно когда классы в признаковом пространстве близки к многомерному нормальному распределению и ковариации корректно оценены. SAM показывает средние результаты (0,82/0,80/0,79;  $\kappa=0,78$ ): метод устойчив к вариациям освещённости, поскольку использует углы между спектральными векторами, но хуже различает классы с похожей формой спектра (например, пастбища и пашня) и не учитывает дисперсию/ковариацию. Наименее точен Minimum Distance (0,79/0,76/0,75;  $\kappa=0,72$ ), что ожидаемо: он опирается только на евклидово расстояние до центроидов и игнорирует различия в разбросе признаков.

Если сопоставить группы алгоритмов машинного обучения и параметрических алгоритмов, выигрыш очевиден: при переходе от MLC к RF overall растёт с 0,88 до 0,94 ( $\Delta=+0,06$ ),  $\kappa$  - с 0,84 до 0,91 ( $\Delta=+0,07$ ). Нейросетевые модели также демонстрируют значительный прирост точности: ANN превосходит MLC по user's accuracy, MLP приближается к ансамблевым методам, а CNN фактически достигает уровня CART и SVM. При этом у RF, SVM и CNN выше и producer's, и user's точности, то есть они одновременно лучше восстанавливают истинные пиксели классов и дают более «чистые» пользовательские классы на карте. Практически это означает меньше пропусков природных объектов и меньше смешения классов сельскохозяйственных земель и урбанизированных территорий.

Тем не менее параметрические методы сохраняют ценность как базовый ориентир и инструмент валидации. Для производственных карт LULC в условиях сложного ландшафта рационально отдавать приоритет ансамблевым методам и глубокому обучению (RF, SVM, CNN), тогда как ANN и MLP могут использоваться как промежуточные решения при ограниченных вычислительных ресурсах или небольших наборах данных.

Таким образом, полученные результаты показывают, что, несмотря на достоинства каждого алгоритма, методы машинного и глубокого обучения, особенно CNN, обладают наибольшим потенциалом для высокоточной классификации LULC по сравнению с параметрическими подходами.

## ЗАКЛЮЧЕНИЕ

В представленных методических рекомендациях системно рассмотрены теоретические основы, методологические подходы и практические инструменты классификации землепользования и земельного покрова (Land Use and Land Cover - LULC) на основе данных ДЗЗ и современных геоинформационных технологий. Актуальность данной тематики обусловлена возрастающей потребностью в объективной, оперативной и пространственно детализированной информации о состоянии и динамике земельного покрова в условиях глобальных климатических изменений, усиления антропогенной нагрузки и необходимости рационального управления природными ресурсами. Классификация LULC выступает ключевым инструментом пространственного анализа, позволяющим выявлять закономерности изменения сельскохозяйственных угодий, трансформации ландшафтов, оценивать устойчивость экосистем, контролировать процессы урбанизации, деградации земель, и другие важные социально-экономические и экологические процессы.

В ходе разработки рекомендаций проведён анализ существующих подходов к картографированию землепользования и земельного покрова, включая параметрические методы классификации, непараметрические алгоритмы машинного обучения и современные методы глубокого обучения. Установлено, что традиционные статистические алгоритмы, такие как Maximum Likelihood Classification, Minimum Distance и Spectral Angle Mapper, обладают высокой интерпретируемостью и могут эффективно применяться при ограниченном объёме обучающих данных и близости распределений классов к нормальному. Однако в условиях сложной структуры ландшафта, высокой внутриклассовой вариабельности и спектральной близости объектов их точность существенно снижается. В то же время методы машинного обучения, включая Random Forest, Support Vector Machine и Classification and Regression Trees, демонстрируют более высокую устойчивость к шумам, нелинейность границ решений и способность учитывать взаимодействие

различных признаков, что обеспечивает значительный прирост точности классификации.

Особое внимание уделено перспективам применения методов глубокого обучения, в частности Artificial Neural Network, Multi-Layer Perceptron и Convolutional Neural Network. Показано, что нейросетевые модели способны автоматически извлекать сложные спектральные, текстурные и пространственные признаки из спутниковых изображений, обеспечивая высокую точность классификации даже на гетерогенных территориях и при наличии смешанных пикселей. Использование сверточных нейронных сетей позволяет учитывать пространственный контекст изображений, что существенно повышает качество выделения объектов землепользования и снижает уровень «пиксельного шума» на итоговых картах. Таким образом, интеграция методов машинного и глубокого обучения рассматривается как наиболее перспективное направление развития технологий классификации LULC.

В рекомендациях подробно рассмотрены этапы подготовки исходных данных, включая выбор спутниковых систем, предварительную обработку изображений, расчет спектральных индексов и формирование обучающих выборок. Подчеркивается, что качество исходных данных и репрезентативность обучающей выборки являются критическими факторами, определяющими точность конечных результатов. Рассмотрены подходы к оценке точности классификации, включая использование матрицы ошибок, показателей Overall Accuracy, Producer's Accuracy, User's Accuracy и коэффициента согласия Каппа, что обеспечивает объективную оценку эффективности применяемых методов.

Практическая значимость разработанных рекомендаций заключается в возможности их применения в широком спектре задач пространственного анализа и управления территориями. Результаты классификации LULC могут использоваться для мониторинга состояния окружающей среды, оценки эффективности природоохранных мероприятий, планирования сельскохозяйственного производства, анализа урбанизации, разра-

ботки стратегий устойчивого развития и рационального природопользования. Полученные карты землепользования и земного покрова могут служить информационной основой для принятия управленческих решений на региональном и национальном уровнях, а также для научных исследований в области географии, экологии, аграрных наук и градостроительства.

Важным направлением применения рекомендаций является использование облачных платформ обработки пространственных данных, таких как Google Earth Engine, которые обеспечивают доступ к глобальным архивам спутниковых изображений и позволяют автоматизировать процессы анализа и картографирования. Это открывает новые возможности для оперативного мониторинга изменений земного покрова, анализа многовременных рядов данных и создания динамических систем наблюдения за состоянием территорий.

Перспективы дальнейшего развития методов классификации LULC связаны с интеграцией данных различных источников, включая спутниковые системы высокого пространственного разрешения, беспилотные летательные аппараты, наземные наблюдения и социально-экономические данные. Особое значение имеет развитие методов глубокого обучения, гибридных моделей и автоматизированных алгоритмов обработки больших данных, что позволит повысить точность, масштабируемость и оперативность классификационных работ. Также актуальной задачей является создание национальных и региональных систем мониторинга землепользования, обеспечивающих регулярное обновление карт и анализ пространственно-временной динамики изменений.

Таким образом, разработанные методические рекомендации формируют научно обоснованную и практикоориентированную основу для проведения работ по классификации землепользования и земного покрова. Их внедрение позволит повысить качество пространственного анализа, обеспечить эффективное управление земельными ресурсами и природной средой, а также способствовать достижению целей устойчивого развития. В условиях цифровой трансформации и роста

доступности данных дистанционного зондирования совершенствование методов классификации LULC становится одним из ключевых направлений развития геоинформационных технологий и пространственного анализа.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Richards J.A., Jia X. Remote Sensing Digital Image Analysis. — Springer, 2006. DOI: <https://doi.org/10.1007/3-540-29711-1>
2. Congalton R.G., Green K. Assessing the Accuracy of Remotely Sensed Data. — CRC Press, 2019. DOI: <https://doi.org/10.1201/9780429052729>
3. Foody G.M. Status of land cover classification accuracy assessment. Remote Sensing of Environment, 2002. DOI: [https://doi.org/10.1016/S0034-4257\(01\)00295-4](https://doi.org/10.1016/S0034-4257(01)00295-4)
4. Olofsson P. et al. Good practices for estimating area and assessing accuracy of land change. Remote Sensing of Environment, 2014. DOI: <https://doi.org/10.1016/j.rse.2014.02.015>
5. Breiman L. Random Forests. Machine Learning, 2001. DOI: <https://doi.org/10.1023/A:1010933404324>
6. Cortes C., Vapnik V. Support-vector networks. Machine Learning, 1995. DOI: <https://doi.org/10.1007/BF00994018>
7. Krizhevsky A., Sutskever I., Hinton G.E. ImageNet classification with deep convolutional neural networks. NIPS, 2012. DOI: <https://doi.org/10.1145/3065386>
8. LeCun Y., Bengio Y., Hinton G. Deep learning. Nature, 2015. DOI: <https://doi.org/10.1038/nature14539>
9. Zhu X.X. et al. Deep Learning in Remote Sensing. IEEE Geoscience and Remote Sensing Magazine, 2017. DOI: <https://doi.org/10.1109/MGRS.2017.2762307>
10. Ma L. et al. Deep learning in remote sensing applications. ISPRS Journal of Photogrammetry and Remote Sensing, 2019. DOI: <https://doi.org/10.1016/j.isprsjprs.2019.04.015>
11. Pelletier C. et al. Temporal convolutional neural networks for land cover classification. Remote Sensing, 2019. DOI: <https://doi.org/10.3390/rs11050523>
12. Belgiu M., Drăguț L. Random Forest in remote sensing. ISPRS Journal of Photogrammetry and Remote Sensing, 2016. DOI: <https://doi.org/10.1016/j.isprsjprs.2016.01.011>
13. Maxwell A.E. et al. Implementation of machine-learning classification in remote sensing. ISPRS Journal of Photogrammetry and Remote Sensing, 2018. DOI: <https://doi.org/10.1016/j.isprsjprs.2018.09.018>
14. Pekel J.F. et al. High-resolution mapping of global surface water. Nature, 2016. DOI: <https://doi.org/10.1038/nature20584>
15. Hansen M.C. et al. Global forest change. Science, 2013. DOI: <https://doi.org/10.1126/science.1244693>
16. Friedl M.A. et al. MODIS global land cover mapping. Remote Sensing of Environment, 2010. DOI: <https://doi.org/10.1016/j.rse.2009.08.016>
17. Wulder M.A. et al. Landsat continuity. Remote Sensing of Environment, 2012. DOI: <https://doi.org/10.1016/j.rse.2011.09.011>

18. Gorelick N. et al. Google Earth Engine: Planetary-scale geospatial analysis. *Remote Sensing of Environment*, 2017. DOI: <https://doi.org/10.1016/j.rse.2017.06.031>
19. Li W. et al. Deep learning-based oil palm classification. *Remote Sensing of Environment*, 2017. DOI: <https://doi.org/10.1016/j.rse.2017.05.010>
20. Zhang L. et al. Deep learning for remote sensing data. *IEEE Geoscience and Remote Sensing Magazine*, 2016. DOI: <https://doi.org/10.1109/MGRS.2016.2540798>
21. Tatem A.J. et al. Fifty years of Earth observation satellites. *Bioscience*, 2008. DOI: <https://doi.org/10.1641/B580201>
22. Pontius R.G., Millones M. Death to Kappa. *International Journal of Remote Sensing*, 2011. DOI: <https://doi.org/10.1080/01431161.2011.552923>
23. Huete A.R. A Soil-Adjusted Vegetation Index (SAVI). *Remote Sensing of Environment*, 1988. DOI: [https://doi.org/10.1016/0034-4257\(88\)90106-X](https://doi.org/10.1016/0034-4257(88)90106-X)
24. Tucker C.J. Red and photographic infrared combinations for vegetation monitoring. *Remote Sensing of Environment*, 1979. DOI: [https://doi.org/10.1016/0034-4257\(79\)90013-0](https://doi.org/10.1016/0034-4257(79)90013-0)
25. Hansen M.C., Loveland T.R. A review of large area monitoring of land cover change. *Remote Sensing of Environment*, 2012. DOI: <https://doi.org/10.1016/j.rse.2011.08.024>
26. Pal M., Mather P.M. Support vector machines for classification in remote sensing. *International Journal of Remote Sensing*, 2005. DOI: <https://doi.org/10.1080/01431160512331314083>
27. Foody G.M., Mathur A. Toward intelligent training of supervised image classifications. *Remote Sensing of Environment*, 2004. DOI: <https://doi.org/10.1016/j.rse.2004.02.003>
28. Rodriguez-Galiano V.F. et al. An assessment of the effectiveness of random forest for land-cover classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2012. DOI: <https://doi.org/10.1016/j.isprsjprs.2011.11.002>
29. Mountrakis G., Im J., Ogole C. Support vector machines in remote sensing. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2011. DOI: <https://doi.org/10.1016/j.isprsjprs.2010.11.001>
30. Gislason P.O., Benediktsson J.A., Sveinsson J.R. Random forests for land cover classification. *Pattern Recognition Letters*, 2006. DOI: <https://doi.org/10.1016/j.patrec.2006.12.003>
31. Camps-Valls G., Bruzzone L. Kernel-based methods for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 2005. DOI: <https://doi.org/10.1109/TGRS.2005.846154>
32. Li S., Dragicevic S., Castro F.A. Geospatial analysis using deep learning. *Computers, Environment and Urban Systems*, 2019. DOI: <https://doi.org/10.1016/j.compenvurbsys.2019.101373>
33. Zhu X.X. et al. Deep learning for Earth observation. *IEEE Geoscience and Remote Sensing Magazine*, 2017. DOI: <https://doi.org/10.1109/MGRS.2017.2762307>

34. Ball J.E., Anderson D.T., Chan C.S. Comprehensive survey of deep learning in remote sensing. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2017. DOI: <https://doi.org/10.1109/JSTARS.2017.2692214>

35. Ma L. et al. A review of supervised object-based land-cover classification. ISPRS Journal of Photogrammetry and Remote Sensing, 2017. DOI: <https://doi.org/10.1016/j.isprsjprs.2017.07.001>

## ГЛОССАРИЙ

**Агроландшафт** - природно-антропогенная территориальная система, сформированная в результате сельскохозяйственной деятельности человека и включающая пашни, пастбища, сенокосы, сельскохозяйственную инфраструктуру и сопутствующие природные элементы.

**Агрегирование данных** - процесс объединения пространственных данных различного уровня детализации в укрупнённые категории или ячейки для упрощения анализа и визуализации.

**Алгоритм классификации** - формализованная процедура отнесения пикселей или объектов к определённым классам землепользования или земного покрова на основе их признаков.

**Алгоритмы глубокого обучения** - методы машинного обучения, основанные на многослойных нейронных сетях, способные автоматически извлекать сложные иерархические признаки из данных.

**Анализ пространственных данных** - совокупность методов обработки, интерпретации и моделирования географически привязанных данных с целью выявления закономерностей и взаимосвязей.

**Базовая карта** - картографическая основа, используемая для отображения тематических данных, включая границы территорий, гидрографию, дорожную сеть и другие ориентиры.

**База пространственных данных** - структурированная совокупность географических данных, организованных для хранения, обработки и анализа в геоинформационных системах.

**Бинарная классификация** - классификация, при которой объекты распределяются между двумя классами, например «растительность» и «нерастительность».

**Валидация классификации** - процесс проверки точности результатов классификации путём сравнения с эталонными данными или полевыми наблюдениями.

**Вегетационные индексы** - числовые показатели, вычисляемые на основе спектральных каналов спутниковых данных для оценки состояния растительности.

**Векторные данные** - пространственные данные, представленные в виде точек, линий и полигонов.

**Временной ряд спутниковых данных** - последовательность спутниковых изображений одной территории, полученных в разные моменты времени.

**Геоинформационная система (ГИС)** - аппаратно-программный комплекс для работы с пространственными данными, включающий средства их хранения, анализа и визуализации.

**Геопривязка** - процесс сопоставления изображения или карты с координатной системой Земли.

**Гетерогенный ландшафт** - территория с высокой пространственной неоднородностью, состоящая из различных типов землепользования и покрова.

**Данные дистанционного зондирования Земли (ДЗЗ)** - информация о поверхности Земли, полученная с помощью спутниковых или авиационных сенсоров.

**Деградация земель** - ухудшение состояния почв и растительности под воздействием природных и антропогенных факторов.

**Дерево решений** - алгоритм машинного обучения, представляющий процесс классификации в виде последовательности логических условий.

**Землепользование (Land Use)** - функциональное использование территории человеком (сельское хозяйство, промышленность, жилые зоны и др.).

**Земной покров (Land Cover)** - физическое покрытие поверхности Земли природными или антропогенными объектами.

**Индекс NDVI** - нормализованный разностный вегетационный индекс, используемый для оценки состояния растительности.

**Индекс NDWI** - индекс, применяемый для выявления водных объектов и влажности.

**Индекс BSI** - индекс оголённой почвы, применяемый для выделения почвенных поверхностей.

**Искусственная нейронная сеть (ANN)** - модель машинного обучения, имитирующая работу биологических нейронных сетей.

**Классификация изображений** - процесс присвоения пикселям спутникового изображения определённых классов.

**Кластеризация** - метод группировки объектов без предварительно заданных классов.

**Коэффициент Каппа** - показатель согласия между классификацией и эталонными данными с учётом случайных совпадений.

**Машинное обучение** - область искусственного интеллекта, изучающая методы построения моделей, обучающихся на данных.

**Матрица ошибок** - таблица, используемая для оценки точности классификации.

**Многослойный перцептрон (MLP)** - тип нейронной сети с несколькими слоями нейронов.

**Непараметрические методы** - методы классификации, не требующие предположений о распределении данных.

**Обучающая выборка** - набор размеченных данных для обучения модели.

**Область интереса (ROI)** - территория, выбранная для анализа.

**Общая точность (Overall Accuracy)** - доля правильно классифицированных пикселей.

**Параметрические методы классификации** - методы, основанные на статистических предположениях о данных.

**Пиксель** - минимальный элемент цифрового изображения.

**Предобработка изображений** - этап подготовки спутниковых данных к анализу.

**Растровые данные** - пространственные данные в виде сетки пикселей.

**Random Forest** - ансамблевый алгоритм машинного обучения.

**Сверточная нейронная сеть (CNN)** - модель глубокого обучения для анализа изображений.

**Support Vector Machine (SVM)** - алгоритм машинного обучения для классификации.

**Точность производителя** - вероятность правильного распознавания объектов определённого класса.

**Точность пользователя** - надёжность классифицированных данных для пользователя.

**Устойчивое развитие** - концепция сбалансированного социально-экономического и экологического развития.

**Эталонные данные** - данные, используемые для проверки точности классификации.

*Учебное издание*

**РЕКОМЕНДАЦИИ  
ПО КЛАССИФИКАЦИИ  
ЗЕМЛЕПОЛЬЗОВАНИЯ И ЗЕМЕЛЬНОГО  
ПОКРОВА  
(LAND USE AND LAND COVER)**

Подписано в печать 2.09.2026.  
Формат 60х84 <sup>1</sup>/<sub>16</sub>.  
Печать цифровая. Объем 12,25 п.л.  
Тираж 50 экз.

Отпечатано в типографии «ESKoPrint».

ISBN 978-601-04-7644-8



9|786010|476448|