Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления
КАФЕДРА Теоретическая информатика и компьютерные технологии

КУРСОВАЯ РАБОТА

НА ТЕМУ:

Сравнительный анализ классических и нейросетевых методов повышения разрешения применительно к задаче обработки спутниковых снимков Земли

Студент	ИУ9-51Б		Амбросий Н.Е.
	(группа)	(подпись, дата)	(И.О. Фамилия)
Руководитель курсовой работы			Брагин А.В.
		(подпись, дата)	(И.О. Фамилия)
Консультант			
		(подпись, дата)	(И.О. Фамилия)

2025 г.

Оглавление

ВВЕДЕНИЕ

Интенсивное развитие спутниковых технологий и вычислительных систем превратило данные дистанционного зондирования Земли (ДЗЗ) в фундаментальный инструмент для решения широкого спектра глобальных и прикладных задач. Спутниковые снимки лежат в основе систем мониторинга экологической обстановки, точного земледелия, контроля за разработкой недр, урбанистики и управления территориями. Эффективность этих систем напрямую зависит от пространственного разрешения получаемых изображений, поскольку именно детализация определяет возможность достоверной идентификации объектов, оценки их состояния и динамики изменений. Однако дальнейшее повышение разрешающей способности съемочной аппаратуры космических аппаратов сталкивается с фундаментальными физическими ограничениями, такими как дифракционный предел оптики, и экономическими барьерами, связанными с высокой стоимостью разработки, производства и вывода на орбиту спутников нового поколения.

В этих условиях особую актуальность приобретают программные методы повышения разрешения изображений, известные как технологии сверхразрешения (Super-Resolution, SR). Данные подходы позволяют алгоритмически восстанавливать высокочастотную информацию из одного или нескольких снимков низкого разрешения, преодолевая аппаратные ограничения без необходимости модернизации спутниковой группировки. Исторически для этих целей применялись классические интерполяционные методы, обеспечивающие предсказуемый результат, но часто приводящие к размытию границ и потере мелких текстур. С развитием вычислительных мощностей доминирующую позицию стали занимать нейросетевые подходы, способные генерировать визуально более четкие и детализированные изображения. Однако специфика ДЗЗ, включающая сложные текстуры и наличие атмосферных искажений, делает прямое применение существующих SR-моделей нетривиальной задачей, так как они могут вносить артефакты, недопустимые при точном дешифрировании.

Данная работа ставит своей целью проведение сравнительного анализа эффективности классических и нейросетевых методов повышения разрешения применительно к задаче обработки спутниковых снимков Земли. В рамках исследования последовательно решается несколько задач: выполняется теоретический анализ существующих подходов, разрабатываются алгоритмы для подготовки данных и их последующей обработки с применением рассматриваемых методов. В работе проводится сравнительный эксперимент, где эффективность алгоритмов оценивается с использованием количественных метрик и визуального анализа. На основе полученных результатов формулируются итоговые выводы о преимуществах, недостатках и границах применимости классических и нейросетевых подходов для решения задач, связанных с обработкой спутниковых изображений.

1. Теоретические основы задачи Super-Resolution

1.1 Математическая постановка задачи

Задача повышения разрешения изображений относится к классу некорректно поставленных обратных задач (ill-posed inverse problems). Основной целью является восстановление высокочастотных деталей изображения высокого разрешения (High Resolution, HR) на основе имеющегося изображения низкого разрешения (Low Resolution, LR).

Пусть $I_{HR} \in \mathbb{R}^{H \times W \times C}$ — исходное изображение высокого разрешения, где H, W и C — его высота, ширина и количество каналов соответственно.

Процесс формирования наблюдаемого изображения низкого разрешения $I_{LR} \in \mathbb{R}^{h \times w \times C}$(где h = H/s, w = W/s, a s — коэффициент масштабирования) описывается моделью деградации. Данная модель имитирует потерю качества, происходящую при съемке аппаратурой спутника из-за ограничений оптики и сенсоров. В общем виде связь между I_{LR} и I_{HR} выражается следующим образом:

$$I_{LR} = D(I_{HR}; \delta) \quad (1)$$

Где D — оператор деградации, зависящий от δ (масштабирование, размытие, шум).

Для проведения численных экспериментов общая модель деградации D конкретизируется. В качестве оператора понижения разрешения используется стандартный алгоритм бикубической интерполяции. Таким образом, обучающая выборка формируется из пар изображений, где эталонное изображение высокого разрешения I_{HR} служит для генерации соответствующего ему изображения низкого разрешения I_{LR} по следующей модели:

$$I_{LR} = Bicubic(I_{HR}, s) \quad (2)$$

Целью алгоритма повышения разрешения является нахождение функции отображения F, которая позволяет получить восстановленное изображение I_{SR} из I_{LR}, максимально приближенное к исходному I_{HR}:

$$I_{SR} = F(I_{LR}) \approx I_{HR} \quad (3)$$

1.2 Неглубокие методы повышения разрешения

Классические подходы к задаче Super-Resolution базируются на методах интерполяции. Основной принцип — оценка значений пикселей изображения высокого разрешения на основе локальной окрестности пикселей исходного изображения низкого разрешения. В рамках данной работы рассматриваются и программно реализуются два наиболее распространенных метода: бикубическая интерполяция и интерполяция Ланцоша.

Бикубическая интерполяция является усовершенствованием билинейного метода и обеспечивает более гладкие переходы яркости, сохраняя при этом больше деталей. Метод аппроксимирует значение пикселя с использованием полиномов третьей степени.

Для вычисления значения нового пикселя рассматривается окрестность размером 4 x 4 пикселя исходного изображения. Вклад каждого соседа определяется весовой функцией W(d), зависящей от расстояния d до искомой точки.

В работе алгоритм основан на стандартном кубическом ядре свертки. Функция весов W(d) задается следующей кусочно-заданной функцией:

$$W(d)=\begin{cases}(a+2)|d^3|-(a+3)|d^2|+1 & |d|\le 1\\ a|d^3|-5a|d^2|+8a|d|-4a & 1<|d|<2\\ 0 & \text{else}\end{cases} \quad (4)$$

Где a — свободный параметр, определяющий характер интерполяции (резкость). Обычно принимается значение a = -0.5.

Другим подходом к интерполяции является фильтр Ланцоша. Данный метод позволяет достичь высокой четкости границ объектов за счет использования весовой функции, основанной на нормированном кардинальном синусе, или функции sinc(x):

$$\text{sinc}(x)=\frac{\sin(\pi x)}{\pi x}$$

Ядро фильтра Ланцоша L(x) формируется на основе функции sinc(x). Чтобы локализовать ее влияние в пределах заданной окрестности, она умножается на оконную функцию, в качестве которой выступает масштабированная версия самой sinc-функции. Формула для вычисления весовых коэффициентов имеет следующий вид:

$$L(x)=\begin{cases}\operatorname{sinc}(x)\cdot\operatorname{sinc}(\frac{x}{a}) & -a<x<a \\ 0 & \text{else}\end{cases} \quad (5)$$

Где параметр a определяет размер окна фильтра(в даной работе взято a=3).

1.2 Нейросетевые методы повышения разрешения

В отличие от интерполяционных подходов, нейросетевые модели способны изучать закономерности и текстуры на больших наборах данных и затем использовать эти знания для восстановления высокочастотных деталей, утраченных при понижении разрешения.

Одной из первых работ, продемонстрировавших эффективность такого подхода, стала архитектура SRCNN (Super-Resolution Convolutional Neural Network)[1]. Ее структура состоит из трех последовательных сверточных слоев, отвечающих за извлечение признаков, нелинейное отображение и финальную реконструкцию. Недостатком SRCNN является то, что вся обработка выполняется на предварительно увеличенном с помощью бикубической интерполяции изображении.

Дальнейшим развитием этого подхода стала архитектура FSRCNN (Fast Super-Resolution Convolutional Neural Network)[2], которая и была реализована в рамках данной работы. Архитектурные различия между SRCNN и FSRCNN проиллюстрированы на рисунке 1.1. В новой модели перенесли основную вычислительную нагрузку в пространство низкого разрешения, что повышает ее быстродействие. Сеть принимает на вход исходное LR-изображение без предварительного увеличения, а операция апскейлинга выполняется однократно на финальном этапе с помощью слоя транспонированной свертки. Была переработана внутренняя структура сети: вместо одного слоя нелинейного отображения, как в SRCNN, FSRCNN использует более сложную последовательность, включающую сужающий слой для уменьшения размерности признаков, несколько отображающих слоев и расширяющий слой для возврата к исходной размерности. В архитектуре применяются сверточные ядра меньшего размера при одновременном увеличении общей глубины сети.

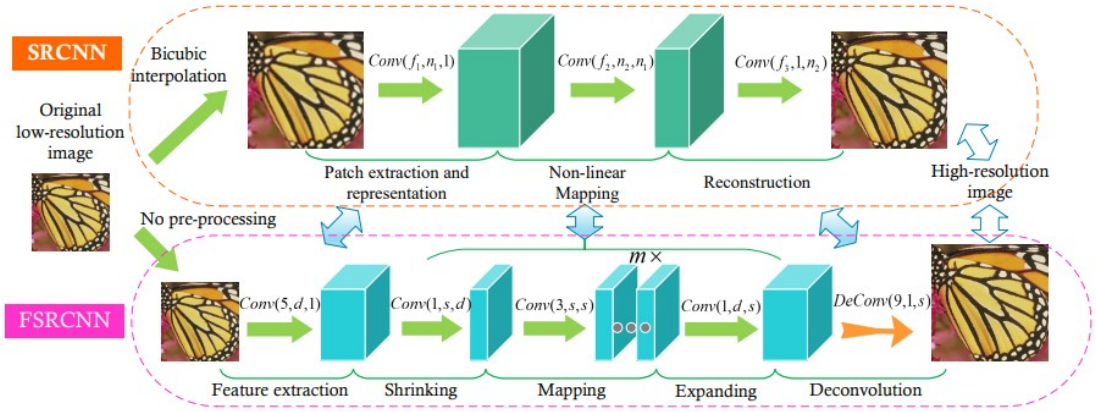


Рисунок 1.1 — Сравнение архитектур SRCNN и FSRCNN. Источник: Dong C. et al., 2016

Сравнительно неглубокая структура FSRCNN накладывает ограничения на выразительную способность модели. Прямое увеличение количества слоев приводит к проблеме затухания градиентов, что затрудняет обучение. Решением этой проблемы стало применение концепции остаточного обучения (residual learning), впервые предложенной в архитектуре ResNet.

В данной работе реализована архитектура, далее именуемая ResSR, которая использует данный подход. Ее работа начинается с начального сверточного слоя, который преобразует входное LR-изображение в пространство признаков с большей размерностью. Затем эти признаки поступают в основной модуль, состоящий из последовательности остаточных блоков (ResidualBlock). Каждый такой блок содержит два сверточных слоя и пробрасываемое соединение (skip connection), суммирующее вход блока с его выходом, что позволяет сети обучаться остаточной функции и упрощает оптимизацию. Помимо этих локальных соединений, в архитектуре также реализовано глобальное остаточное обучение: признаки, полученные после первого слоя, складываются с признаками, прошедшими через всю последовательность остаточных блоков. После извлечения глубоких признаков выполняется операция апскейлинга. На заключительном этапе финальный сверточный слой реконструирует из этих признаков итоговое HR-изображение.

Еще один вариант глубокой сети - архитектура RCAN (Residual Channel Attention Network)[3]. Вместо стандартного остаточного блока используется RCAB (Residual Channel Attention Block).

Основу RCAB составляет канальный механизм внимания (Channel Attention). Его задача — явно смоделировать взаимозависимости между каналами в пространстве признаков. Для этого механизм сначала агрегирует пространственную информацию каждого канала с помощью глобального среднего пулинга, получая вектор, который описывает текущее состояние всех признаков. Затем этот вектор обрабатывается небольшой нейронной сетью, которая на выходе генерирует набор весовых коэффициентов — по одному для каждого канала. Эти коэффициенты, нормализованные в диапазоне от 0 до 1, отражают относительную важность каждого признака в данный момент. На последнем этапе исходные карты признаков умножаются на эти вычисленные веса.

1.3. Метрики оценки качества восстановления

Для объективного сравнительного анализа реализованных алгоритмов необходимо использование количественных метрик качества, вычисляемых путем сравнения восстановленного изображения с эталонным изображением высокого разрешения. В данной работе используются три метрики: PSNR, SSIM, ERGAS

Метрика PSNR (Peak Signal-to-Noise Ratio) является наиболее распространенным инструментом оценки качества реконструкции изображений. Она определяет отношение максимально возможной энергии сигнала к энергии шума, вносимого искажениями алгоритма обработки.

PSNR выражается в децибелах (дБ) и вычисляется через среднеквадратичную ошибку (MSE):

$$\mathrm{MSE} = \frac{1}{H \cdot W \cdot C} \sum_{i=1}^{H} \sum_{j=1}^{W} \sum_{k=1}^{C} \left(I_{HR}(i,j,k) - I_{SR}(i,j,k) \right)^2$$

$$\mathrm{PSNR} = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{\mathrm{MSE}} \right) \quad (6)$$

где H, W — размеры изображения, C — количество каналов, MAX_I — максимальное возможное значение пикселя.

Чем выше значение PSNR, тем меньше искажений внесено при восстановлении. Однако данная метрика плохо коррелирует с восприятием человеческим глазом и структурной четкостью, часто отдавая предпочтение размытым изображениям перед резкими, но имеющими небольшие пиксельные отклонения.

Для оценки сохранения структурной информации, что критически важно для дешифрирования объектов на спутниковых снимках, используется метрика SSIM (Structural Similarity Index). В отличие от PSNR, она учитывает изменения яркости, контраста и структуры.

SSIM между двумя окнами x и y изображений вычисляется по формуле:

$$\mathrm{SSIM}(x,y)=\frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2+\sigma_y^2+c_2)} \quad (7)$$

где:

μ_x, μ_y — средние значения яркости в окнах;

σ_x^2, σ_y^2 — дисперсии (оценка контраста);

σ_{xy} — ковариация (оценка структуры);

c_1, c_2 — константы для стабилизации деления при малых знаменателях.

Итоговое значение метрики лежит в диапазоне [-1, 1], где 1 означает полную идентичность изображений.

Поскольку работа ведется со спутниковыми данными, целесообразно использование специализированной метрики ERGAS (Relative dimensionless global error synthesis), разработанной для оценки качества слияния и восстановления мультиспектральных изображений.

Данная метрика оценивает глобальное качество синтезированного изображения с учетом коэффициента масштабирования:

$$\mathrm{ERGAS}=\frac{100\cdot h}{l}\sqrt{\frac{1}{C}\sum_{k=1}^{C}\left(\frac{\mathrm{RMSE}_k}{\mu_k}\right)^2} \quad (8)$$

Где:

h/l — отношение пространственного разрешения панхроматического и мультиспектрального сенсоров (в контексте SR это коэффициент масштабирования l/s);

$RMSE_k$ — среднеквадратичная ошибка для k-го канала;

μ_k — средняя яркость k-го канала эталонного изображения.

Чем меньше значение ERGAS, тем выше спектральное и пространственное качество восстановленного снимка.

2. Программная реализация и методика проведения эксперимента

2.1 Программная реализация классических методов

2.1.1. Бикубическая интерполяция

Программная реализация алгоритма бикубической интерполяции находитс\ в функции SR_bicubic, которая принимает на вход изображение низкого разрешения и целевые размеры, а возвращает изображение высокого разрешения. Логика ее работы состоит из нескольких последовательных этапов.

Первоначально выполняется подготовка входного изображения. Поскольку для вычисления значения каждого нового пикселя требуется окрестность размером 4x4, к границам исходного изображения добавляется отступ.

Далее определяются координаты будущих пикселей в системе отсчета исходного изображения. Вычисление значений пикселей вынесено в отдельную функцию get_bicubic_weights. Она является прямой программной реализацией кусочно-заданной полиномиальной функции, представленной в формуле (4), и возвращает четыре весовых коэффициента для каждого пикселя.

Далее используется свойство сепарабельности бикубического фильтра, что позволяет повысить вычислительную эффективность. Вместо того чтобы для каждого пикселя выполнять двумерную свертку с ядром 4x4, процесс разделяется на две последовательные одномерные интерполяции: сначала вдоль строк, а затем вдоль столбцов. На первом этапе для каждой строки нового изображения выбираются четыре ближайшие строки из исходного изображения. Затем вычисляется их взвешенная сумма с использованием весов W_rows, полученных ранее. Это создает промежуточное изображение, интерполированное только по вертикали. На втором этапе процедура повторяется для столбцов: промежуточное изображение транспонируется, и к его новым строкам (бывшим столбцам) применяется та же логика с весами W_cols. После второго взвешенного суммирования и обратного транспонирования формируется итоговое изображение. Данный двушаговый процесс проиллюстрирован на рисунке 2.1.

```
W_rows = get_bicubic_weights(row_frac)
W_cols = get_bicubic_weights(col_frac)

# choose indices of the 4 neighboring pixels (4 rows and 4 columns)
row_idx = np.stack( arrays: [row_floor - 1, row_floor, row_floor + 1, row_floor + 2], axis=0)
col_idx = np.stack( arrays: [col_floor - 1, col_floor, col_floor + 1, col_floor + 2], axis=0)

# (4, new_height, W+4, C) scaled matrix.
rows_subset = img_padded[row_idx, :, :]

# W_rows: (4, new_height) → (4, new_height, 1, 1)
W_rows_exp = W_rows[:, :, None, None]

# squeeze these 4 rows into one with weighted sum → (new_height, W+4, C)
intermediate = np.sum(rows_subset * W_rows_exp, axis=0)

# transpose matrix and repeat for columns
intermediate = intermediate.transpose(1, 0, 2)
cols_subset = intermediate[col_idx, :, :]
W_cols_exp = W_cols[:, :, None, None]
result_transposed = np.sum(cols_subset * W_cols_exp, axis=0)
result = result_transposed.transpose(1, 0, 2)
```

Рисунок 2.1 — Реализация сепарабельной бикубической интерполяции

2.1.2 Интерполяция Ланцоша

Программная реализация метода интерполяции Ланцоша, представленная функцией SR_lanczos, во многом следует архитектуре, использованной для бикубического метода. Алгоритм также основан на принципе сепарабельности, что позволяет разделить двумерную фильтрацию на две последовательные одномерные операции — сначала по строкам, затем по столбцам.

Функция get_lanczos_weights вычисляет весовые коэффициенты на основе ядра Ланцоша, определенного в формуле (5). Вспомогательная функция sinc реализует вычисление нормированного кардинального синуса, корректно обрабатывая случай, когда аргумент равен нулю. Функция get_lanczos_weights итерируется по всем пикселям в пределах окна, размер которого задается параметром a (в данной работе a=3, что означает учет 6 ближайших соседей по каждой оси). Для каждого соседа вычисляется вес.

Основная функция SR_lanczos повторяет логику SR_bicubic. На первом этапе выполняется симметричное дополнение границ изображения, причем размер отступа теперь зависит от параметра a, чтобы обеспечить достаточное количество соседних пикселей для расчетов. После определения целочисленных и дробных координат пикселей в новой сетке вызывается get_lanczos_weights для получения весов. Затем, аналогично бикубической интерполяции, выполняется двухэтапный процесс взвешенного суммирования: сначала для строк, что создает промежуточное, вертикально интерполированное изображение, а затем для столбцов транспонированного промежуточного изображения. Финальный этап постобработки идентичен ранее рассмотренной реализации.

2.2 Подготовка данных

В качестве набора данных для обучения моделей и проведения сравнения в данной работе был выбран UC Merced Land Use Dataset[4]. Изображения для набора были отобраны из аэрофотоснимков Национальной картографической программы Геологической службы США (USGS).

Набор данных состоит из 2100 изображений, равномерно распределенных по 21 классу типов землепользования. Каждый класс представлен 100 изображениями. Технические характеристики изображений следующие:

Размер: 256x256 пикселей.

Спектральный диапазон: RGB.

Классы охватывают широкий спектр сцен: от регулярных, антропогенных объектов (аэропорты, здания, дорожные развязки, парковки) до сложных, нерегулярных природных текстур (лес, пляж, река, сельскохозяйственные угодья). Такое разнообразие позволяет провести комплексную оценку способности классических и нейросетевых методов восстанавливать различные типы деталей: четкие прямолинейные границы, повторяющиеся паттерны и стохастические природные текстуры.

Для корректного обучения нейросетевых моделей и проведения объективной оценки их производительности исходный набор данных был разделен на три независимые выборки: обучающую (train), валидационную (val) и тестовую

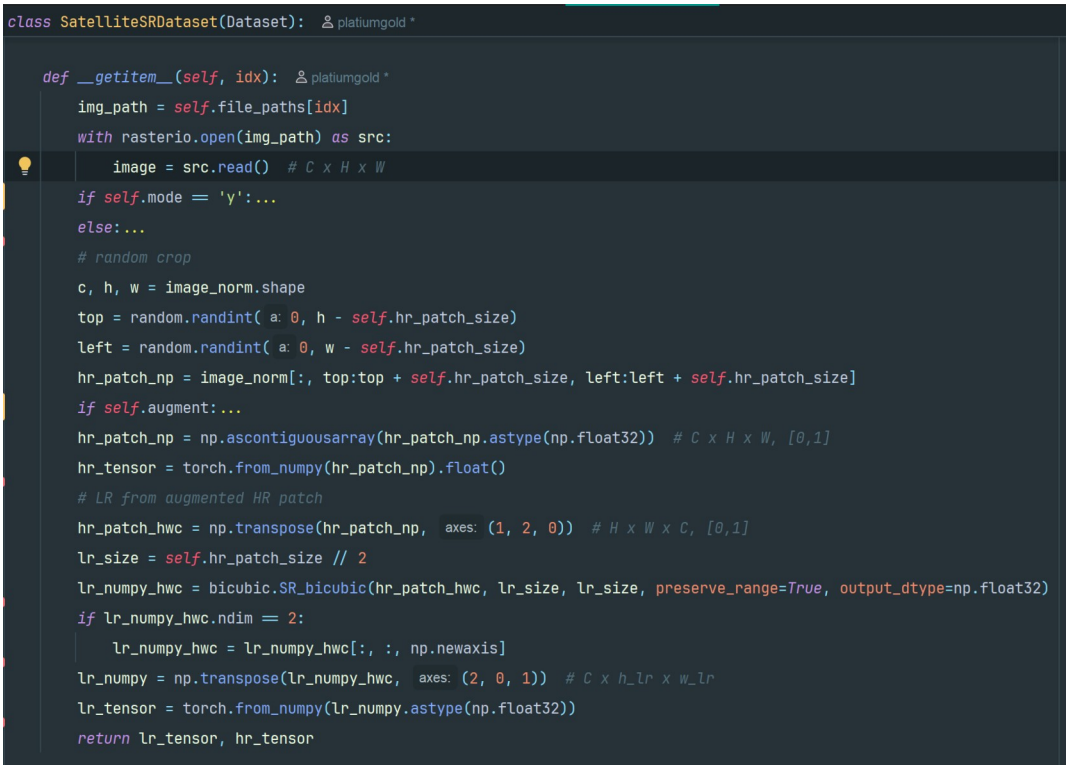
(test). Процесс разделения, реализованный в виде скрипта data_split.py, выполняется итеративно для каждого из 21 классов. Такой подход обеспечивает стратифицированное разделение, при котором исходное процентное соотношение классов сохраняется во всех трех итоговых выборках.

Для разделения списка файлов внутри каждой классовой директории используется функция train_test_split из библиотеки scikit-learn. Процедура разделения гарантирует, что итоговое соотношение выборок составит 80% для обучения (train), 10% для валидации (val) и 10% для тестирования (test). Это достигается в два этапа: сначала из пула изображений каждого класса отбирается 10% для тестового набора, а затем оставшаяся часть делится между обучающей и валидационной выборками в нужной пропорции.

После определения принадлежности каждого файла к одной из выборок скрипт создает новую структуру директорий (DEST_DIR), содержащую подпапки train, val и test. Затем, с помощью функции shutil.copy2 из вспомогательной функции copy_files, файлы физически копируются из исходного местоположения (SOURCE_DIR) в соответствующие целевые папки с сохранением классовой структуры. В результате формируется организованный набор данных, полностью подготовленный для последующих этапов эксперимента.

Для загрузки данных в процессе обучения нейронных сетей был разработан специальный класс SatelliteSRDataset, наследуемый от базового класса torch.utils.data.Dataset. Его основная задача — организация процесса чтения изображений с диска и их предобработка в реальном времени для формирования обучающих пар, состоящих из фрагмента изображения высокого разрешения и его соответствующей версии низкого разрешения.

При инициализации (__init__) класс рекурсивно сканирует указанную директорию и формирует список путей ко всем файлам изображений. Основная логика инкапсулирована в методе __getitem__ (Рисунок 2.2), который вызывается загрузчиком данных для каждого элемента выборки.



```
class SatelliteSRDataset(Dataset):  platiumgold *

    def __getitem__(self, idx):  platiumgold *
        img_path = self.file_paths[idx]
        with rasterio.open(img_path) as src:
            image = src.read()  # C x H x W
        if self.mode == 'y':...
        else:...
        # random crop
        c, h, w = image_norm.shape
        top = random.randint( a: 0, h - self.hr_patch_size)
        left = random.randint( a: 0, w - self.hr_patch_size)
        hr_patch_np = image_norm[:, top:top + self.hr_patch_size, left:left + self.hr_patch_size]
        if self.augment:...
        hr_patch_np = np.ascontiguousarray(hr_patch_np.astype(np.float32))  # C x H x W, [0,1]
        hr_tensor = torch.from_numpy(hr_patch_np).float()
        # LR from augmented HR patch
        hr_patch_hwc = np.transpose(hr_patch_np,  axes: (1, 2, 0))  # H x W x C, [0,1]
        lr_size = self.hr_patch_size // 2
        lr_numpy_hwc = bicubic.SR_bicubic(hr_patch_hwc, lr_size, lr_size, preserve_range=True, output_dtype=np.float32)
        if lr_numpy_hwc.ndim == 2:
            lr_numpy_hwc = lr_numpy_hwc[:, :, np.newaxis]
        lr_numpy = np.transpose(lr_numpy_hwc,  axes: (2, 0, 1))  # C x h_lr x w_lr
        lr_tensor = torch.from_numpy(lr_numpy.astype(np.float32))
        return lr_tensor, hr_tensor
```

Рисунок 2.2 — Реализация метода __getitem__ класса SatelliteSRDataset

Работа этого метода состоит из следующих шагов:

- Чтение изображения. По заданному индексу idx выбирается путь к файлу, и изображение считывается с помощью библиотеки rasterio.
- Нормализация и преобразование. Пиксельные значения приводятся к диапазону [0, 1] путем деления на 255. Если выбран режим 'y', изображение преобразуется из RGB в цветовое пространство YCbCr, и для дальнейшей обработки используется только Y-компонента.
- Случайная вырезка (Random Crop). Из полноразмерного изображения случайным образом вырезается фрагмент (патч) заданного размера (hr_patch_size).
- Аугментация. К вырезанному HR-патчу применяется серия случайных геометрических преобразований: горизонтальные и вертикальные отражения, а также повороты на 90, 180 или 270 градусов. Аугментация делает модель более устойчивой к ориентации объектов на снимках и эффективно расширяет объем обучающих данных.

- Генерация LR-патча. После всех аугментаций над HR-патчем из него генерируется соответствующий LR-патч. Для этого используется ранее реализованная функция бикубической интерполяции SR_bicubic с коэффициентом масштабирования 2.
- Преобразование в тензоры. Подготовленные HR и LR-патчи конвертируются в тензоры PyTorch, после чего они готовы для подачи на вход нейронной сети.

2.3 Архитектуры нейронных сетей

В данном разделе приводится описание программной реализации нейросетевых архитектур, рассмотренных в теоретической части. Все модели были реализованы с использованием фреймворка PyTorch. Описание строится по принципу последовательного усложнения архитектурных решений, от базовой модели FSRCNN до сети с механизмом внимания RCAN.

2.3.1 Архитектура FSRCNN

Модель FSRCNN выполняет основную часть вычислений в пространстве низкого разрешения, что обеспечивает высокое быстродействие. Реализованная в работе архитектура FSRCNN_Y (Рисунок 2.3) предназначена для работы с одноканальными изображениями (канал яркости Y).

```python
class FSRCNN_Y(nn.Module):  platiumgold
    def __init__(self):  platiumgold
        scale_factor = 2
        d, s, m = 56, 12, 4
        channels = 1
        super(FSRCNN_Y, self).__init__()

        self.feature_extraction = nn.Sequential(
            nn.Conv2d(channels, d, kernel_size=5, padding=2), nn.PReLU(d))
        self.shrink = nn.Sequential(
            nn.Conv2d(d, s, kernel_size=1), nn.PReLU(s))

        map_layers = []
        for _ in range(m):
            map_layers.extend([nn.Conv2d(s, s, kernel_size=3, padding=1), nn.PReLU(s)])
        self.mapping = nn.Sequential(*map_layers)

        self.expand = nn.Sequential(
            nn.Conv2d(s, d, kernel_size=1), nn.PReLU(d))
        self.deconv = nn.ConvTranspose2d(d, channels, kernel_size=9, stride=scale_factor, padding=4, output_padding=scale_factor - 1)
```

Рисунок 2.3 — Программная реализация архитектуры FSRCNN

Она состоит из пяти компонент, последовательно преобразующих входной LR-тензор:

- Извлечение признаков (feature_extraction): Первый сверточный слой с ядром 5x5 извлекает из входного изображения начальные признаки.
- Сжатие (shrink): Свертка 1x1 уменьшает количество каналов (глубину) карты признаков, снижая вычислительную сложность последующих операций.
- Отображение (mapping): Несколько последовательных сверточных слоев 3x3 формируют ядро модели, выполняя нелинейное отображение признаков.
- Расширение (expand): Еще одна свертка 1x1 восстанавливает исходное количество каналов, подготавливая признаки к этапу повышения разрешения.
- Транспонированная свертка (deconv): Финальный слой ConvTranspose2d одновременно выполняет апскейлинг карты признаков до целевого разрешения и реконструирует итоговое изображение.
- В качестве функции активации во всех слоях, кроме последнего, используется PReLU (Parametric Rectified Linear Unit).

2.3.2 Архитектура ResSR

Переход к более глубоким сетям реализован в архитектуре ResSR, которая использует концепцию остаточного обучения для борьбы с проблемой затухания градиентов и упрощения процесса оптимизации. Основным строительным элементом сети является остаточный блок ResidualBlock (Рисунок 2.4). Его реализация включает два сверточных слоя 3x3 с функцией активации ReLU между ними. В структуру блока внедрено пробрасываемое соединение (skip connection), которое суммирует входной тензор блока x с результатом работы сверточных слоев. Такая структура позволяет сети напрямую обучаться остаточной функции, то есть разнице между входом и выходом, что стабилизирует обучение глубоких моделей.

```
class ResidualBlock(nn.Module):  2 usages  platiumgold
    def __init__(self, channels):  platiumgold
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(channels, channels, kernel_size=3, padding=1)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(channels, channels, kernel_size=3, padding=1)

    def forward(self, x):  platiumgold
        residual = self.conv2(self.relu(self.conv1(x)))
        return x + residual
```

Рисунок 2.4 — Реализация остаточного блока для архитектуры ResSR

Сама архитектура ResSR состоит из начального сверточного слоя (head), основной части (body), состоящей из последовательности таких остаточных блоков, и конечных слоев для повышения разрешения (upsample) и финальной реконструкции (tail). Кроме того, в модели реализовано глобальное остаточное обучение: результат работы первого слоя x_head складывается с результатом работы всей последовательности остаточных блоков res, что дополнительно улучшает прохождение градиента.

2.3.3 Архитектура RCAN

Архитектура RCAN является дальнейшим развитием идеи остаточного обучения и вводит механизм канального внимания, позволяющий сети адаптивно взвешивать важность различных карт признаков. Это усовершенствование реализовано в виде двух модулей: ChannelAttention и RCAB (Рисунок 2.5).

Модуль ChannelAttention (Рисунок 2.5) реализует логику вычисления весовых коэффициентов для каждого канала. Сначала с помощью глобального усредняющего пулинга пространственная информация каждой карты признаков сжимается до одного числа, формируя вектор-дескриптор. Этот вектор пропускается через небольшую двухслойную нейронную сеть (реализованную с помощью сверток 1x1), которая моделирует взаимозависимости между каналами. Функция активации Sigmoid на выходе нормирует полученные значения в диапазон [0, 1], превращая их в веса. Исходный тензор x затем поэлементно умножается на эти веса.

```
class ChannelAttention(nn.Module):  2 usages  platiumgold *
    def __init__(self, num_features, reduction=16):  platiumgold
        super(ChannelAttention, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(
            nn.Conv2d(num_features, num_features // reduction, kernel_size: 1, bias=True),
            nn.ReLU(inplace=True),
            nn.Conv2d(num_features // reduction, num_features, kernel_size: 1, bias=True),
            nn.Sigmoid()
        )
    def forward(self, x):  platiumgold
        y = self.avg_pool(x)
        y = self.fc(y)
        return x * y
```

Рисунок 2.5 — Реализация механизма канального внимания (ChannelAttention)

Модуль RCAB (Residual Channel Attention Block) (Рисунок 2.6) представляет собой улучшенный остаточный блок. Он имеет ту же базовую структуру, что

и ResidualBlock, но в конец последовательности сверток добавляется модуль ChannelAttention. Таким образом, блок сначала извлекает локальные признаки, а затем перевзвешивает их значимость перед тем, как добавить к исходному сигналу через пробрасываемое соединение. Это позволяет модели фокусироваться на наиболее информативных признаках на каждом этапе обработки.

```
class ChannelAttention(nn.Module):  2 usages  platiumgold *
    def __init__(self, num_features, reduction=16):  platiumgold
        super(ChannelAttention, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(
            nn.Conv2d(num_features, num_features // reduction, kernel_size: 1, bias=True),
            nn.ReLU(inplace=True),
            nn.Conv2d(num_features // reduction, num_features, kernel_size: 1, bias=True),
            nn.Sigmoid()
        )
    def forward(self, x):  platiumgold
        y = self.avg_pool(x)
        y = self.fc(y)
        return x * y

class RCAB(nn.Module):  2 usages  platiumgold
    def __init__(self, channels, reduction=16):  platiumgold
        super(RCAB, self).__init__()
        self.body = nn.Sequential(
            nn.Conv2d(channels, channels, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(channels, channels, kernel_size=3, padding=1),
            ChannelAttention(channels, reduction)
        )
```

Рисунок 2.6 — Реализация остаточного блока с вниманием (RCAB)

2.4. Исследование архитектур и отладка методики

На начальном этапе была предпринята попытка обучить легковесную сеть FSRCNN сразу на полноцветных RGB-изображениях. Однако данный подход не

увенчался успехом: процесс обучения демонстрировал нестабильность, а метрики качества на валидации показывали неудовлетворительные результаты. Был сделан вывод, что неглубокая архитектура не обладает достаточной емкостью для моделирования сложных корреляций между тремя цветовыми каналами одновременно. В связи с этим было решено перейти к классической схеме, предложенной авторами оригинальной статьи: обучение FSRCNN только на канале яркости (Y-канал) пространства YCbCr. Это стабилизировало процесс, но итоговое качество реконструкции лишь немного превысило результаты бикубической интерполяции.

Для корректного сравнения более глубокие архитектуры — ResSR и RCAN — первоначально также были обучены на Y-канале. Уже на этом этапе они продемонстрировали отрыв от FSRCNN, доказав, что глубина сети и наличие остаточных связей являются критическими факторами успеха.

Однако анализ результатов на Y-канале привел к гипотезе, что искусственное ограничение одной компонентой яркости не позволяет глубоким сетям раскрыть свой полный потенциал при работе со сложными спутниковыми текстурами. В отличие от FSRCNN, емкость ResSR и RCAN достаточна для обработки многомерных данных. В ходе последующих экспериментов эти модели были переобучены на RGB-данных. Это дало небольшой прирост в четкости границ и цветопередаче, а также в скорости. Таким образом, был сделан вывод о нецелесообразности тренировки глубоких моделей только на Y-канале.

Параллельно с выбором данных подбиралась функция потерь. Было установлено, что стандартная MSE (L2) приводит к размытию высокочастотных деталей и сглаживанию текстур. Переход на L1 (MAE) позволил повысить визуальную резкость восстанавливаемых изображений. Для обучения модели RCAN была выбрана функция Charbonnier Loss — дифференцируемая аппроксимация L1-метрики, которая с точки зрения итогового качества показала результаты, аналогичные стандартной L1, но является более стабильной при численной оптимизации. В качестве алгоритма оптимизации для всех глубоких сетей был выбран Adam, продемонстрировавший лучшую динамику сходимости по сравнению с классическим SGD.

2.5 Процесс обучения нейронных сетей

Обучение всех нейросетевых моделей проводилось на графическом ускорителе с использованием библиотеки PyTorch. Для подачи данных в модели использовался класс DataLoader, который организует их в батчи и позволяет выполнять предобработку в несколько потоков. Размер батча для всех экспериментов был установлен равным 16.

Общая структура цикла обучения для каждой модели была стандартной. На каждой эпохе модель последовательно обрабатывала все пакеты данных из обучающей выборки (train_loader). Для каждого пакета выполнялся прямой проход, вычислялось значение функции потерь между предсказанным и эталонным изображением, после чего производился обратный проход для вычисления градиентов и шаг оптимизатора для обновления весов модели.

Итоговые гиперпараметры, утвержденные по результатам предварительного исследования, представлены в таблице 2.1.

Таблица 2.1 — Гиперпараметры обучения для реализованных нейросетевых моделей

Модель	**Входные данные**	**Оптимизатор**	**Функция потерь**	**Планировщик скорости обучения**	**Количество эпох**
FSRCNN_Y	Y-канал	SGD	MSELoss	-	2000
ResSR_Y	Y-канал	Adam	L1Loss	ReduceLROnPlateau	200
ResSR_RGB	RGB	Adam	L1Loss	ReduceLROnPlateau	200
RCAN_Y	Y-канал	Adam	CharbonnierLoss	ReduceLROnPlateau	200
RCAN_RGB	RGB	Adam	CharbonnierLoss	ReduceLROnPlateau	200

После завершения каждой эпохи тренировки модель переводилась в режим оценки model.eval(), и на данных из валидационной выборки val_loader, которые модель не видела в процессе обучения, рассчитывалось среднее значение метрики PSNR. Этот показатель служил основной мерой качества модели на данной эпохе.

Для более тонкой настройки глубоких моделей ResSR и RCAN также применялся планировщик скорости обучения ReduceLROnPlateau. Он отслеживал

значение потерь на валидационной выборке и автоматически снижал скорость обучения в два раза, если в течение 10 эпох не происходило ее улучшения.

По завершении процесса обучения с помощью функции loss_psnr_graphic были построены и сохранены графики, визуализирующие динамику изменения функции потерь и метрики PSNR на протяжении всех эпох. Для визуального сравнения с базовыми методами сразу после обучения применялась функция model_to_baseline_compare. Итоговые веса моделей, показавших наилучший результат на валидационной выборке, были сохранены для использования на следующем этапе — проведении всестороннего сравнительного эксперимента.

3. Результаты эксперимента и их анализ

3.1. Количественная оценка качества

Для получения объективной оценки эффективности реализованных алгоритмов был выполнен автоматизированный сценарий тестирования на тестовой части набора данных (210 изображений).

Результаты агрегировались в сводную таблицу (Таблица 3.1), содержащую усредненные значения метрик PSNR, SSIM, ERGAS, а также показатели производительности (время инференса и FPS).

Таблица 3.1 — Сводные показатели эффективности методов

Метод	PSNR (ср.)	PSNR (std)	SSIM (ср.)	SSIM (std)	ERGAS (ср.)	Время (мс)	FPS
Bicubic	31.9303	5.7136	0.8991	0.0719	4.0396	5.16	197.52
Lanczos	32.0095	5.8919	0.8971	0.0764	4.0596	11.23	89.95
y_fsrcnn	31.8138	5.5740	0.8960	0.0728	4.0879	12.90	78.34
y_ressr	32.7866	5.7015	0.9090	0.0680	3.7106	14.44	71.37
y_rcan	32.8937	5.6885	0.9104	0.0665	3.6680	20.31	49.85
rgb_ressr	32.8504	5.6990	0.9099	0.0672	3.6824	8.82	122.97
rgb_rcan	32.9474	5.6816	0.9115	0.0648	3.6407	14.98	68.73

Анализ табличных данных позволяет сделать следующие выводы:

- Лидер по качеству: Наивысшие показатели демонстрирует модель rgb_rcan со средним PSNR 32.95 дБ и SSIM 0.9115. Это подтверждает гипотезу о том, что глубокая архитектура с механизмом внимания и обучением на RGB-данных наиболее эффективно восстанавливает детали.
- Эффективность rgb_ressr: Модель rgb_ressr отстает от лидера всего на 0.1 дБ (32.85 дБ), что является незначительной разницей. При этом, как будет показано далее, она обладает более высокой производительностью.
- Неудача FSRCNN: Модель y_fsrcnn показала средний PSNR 31.81 дБ, что даже ниже, чем у классической бикубической интерполяции (31.93 дБ). Это подтверждает вывод главы 2 о том, что малая глубина сети не позволяет ей выучить признаки, превосходящие по качеству интерполяцию на сложных спутниковых снимках.

- Сравнение классики: Метод Ланцоша (Lanczos) показывает стабильное превосходство над бикубическим методом (+0.07 дБ), но сильно уступает глубоким нейросетям (почти на 1 дБ).

Для наглядного сравнения метрик были построены столбчатые диаграммы (Рисунок 3.1).

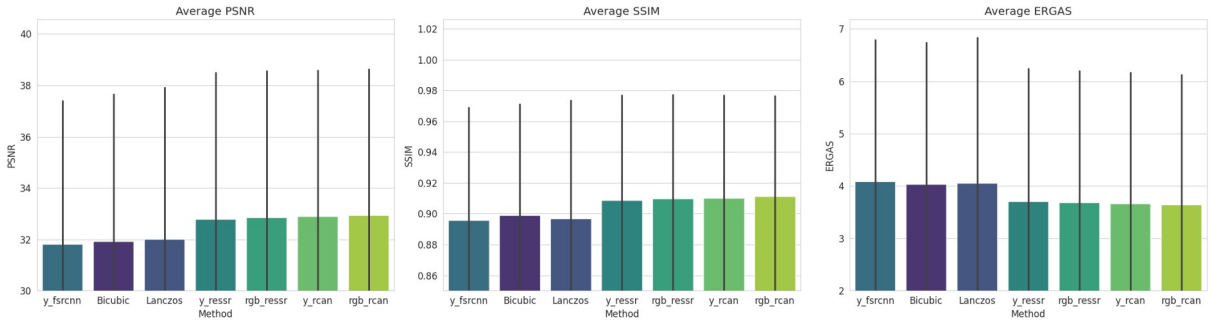


Рисунок 3.1 — Сравнительный анализ средних значений метрик PSNR, SSIM и ERGAS

Графики визуализируют разрыв между группой «слабых» методов (Bicubic, Lanczos, FSRCNN) и группой «сильных» методов (ResSR, RCAN). Метрика ERGAS (ошибка синтеза) у глубоких моделей заметно ниже, что свидетельствует о более точной спектральной и пространственной реконструкции.

Дополнительно была исследована стабильность работы алгоритмов с помощью диаграммы размаха (Рисунок 3.2).

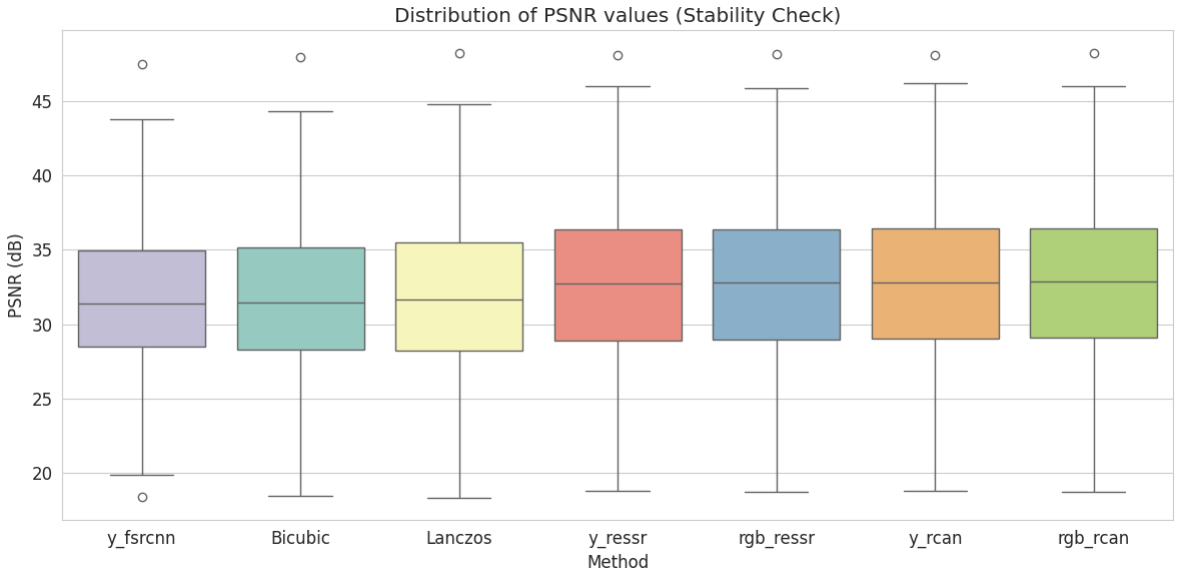


Рисунок 3.2 — Статистическое распределение метрики PSNR

Диаграмма показывает, что глубокие модели (rgb_rcan, rgb_ressr) не просто имеют более высокое среднее, но и «поднимают нижнюю планку» качества. Их минимальные значения находятся выше, чем у классических методов. Это означает, что нейросети гораздо реже допускают грубые ошибки восстановления даже на сложных изображениях.

3.2. Анализ производительности и компромисса «скорость-качество»

Еще одним результатом эксперимента является анализ соотношения качества и скорости работы (FPS), представленный на Рисунке 3.3.

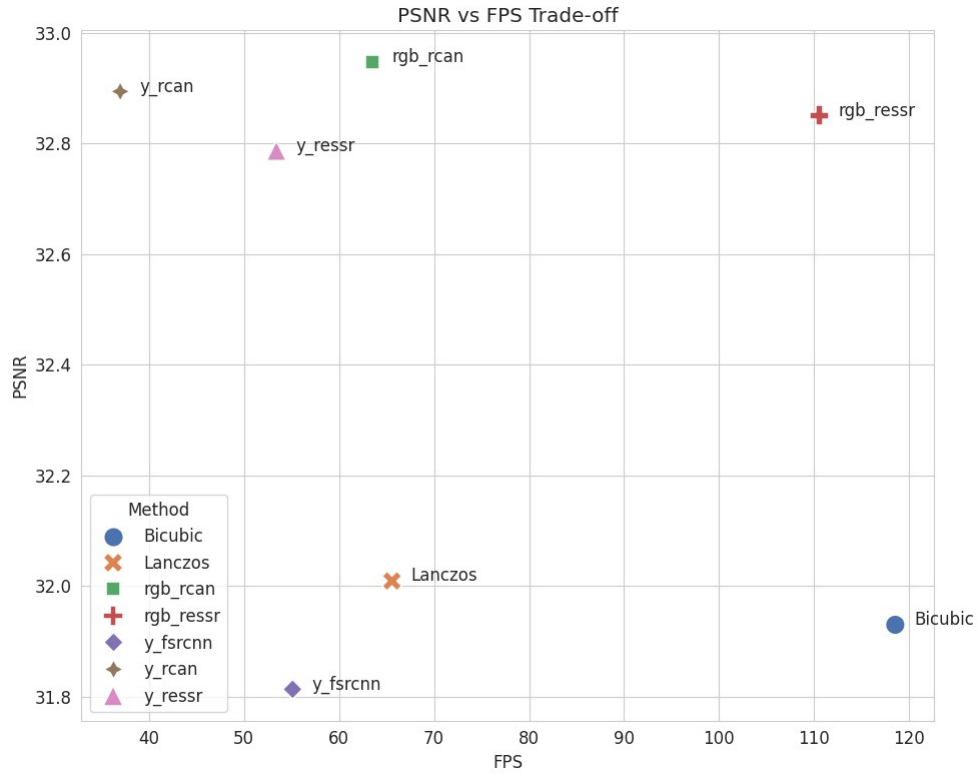


Рисунок 3.3 — Соотношение производительности и качества реконструкции

Этот график позволяет выявить наиболее сбалансированные решения:

rgb_ressr — Оптимальный выбор: Данная модель находится в уникальной позиции (отмечена красным крестом). Она обеспечивает качество, практически идентичное лидеру rgb_rcan (~32.85 дБ), но работает в 2 раза быстрее (~110 FPS

против ~65 FPS у RCAN). Это делает архитектуру ResSR наиболее предпочтительной для реального применения.

y_fsrcnn — Аутсайдер: Модель находится в левом нижнем углу графика. Она проигрывает глубоким сетям по качеству и скорости в данной реализации. Это делает её использование нецелесообразным.

Bicubic — Эталон скорости: Классический метод ожидаемо самый быстрый (~120 FPS), но качество его работы не высоко.

Вывод: Усложнение архитектуры от ResSR к RCAN в данной задаче приводит к падению производительности без существенного выигрыша в качестве.

3.3. Анализ устойчивости к типам местности

Для анализа поведения в разных ситуациях была построена тепловая карта зависимости PSNR от класса землепользования (Рисунок 3.4).

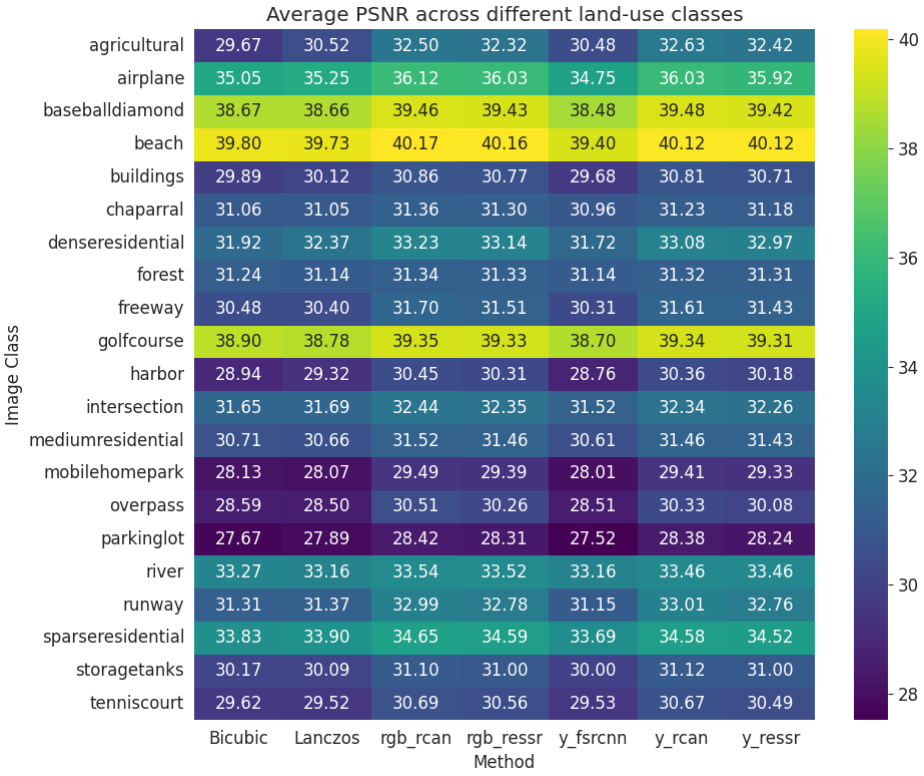


Рисунок 3.4 — Тепловая карта зависимости PSNR от класса землепользования

Анализ тепловой карты позволяет выявить четкую зависимость эффективности методов от семантического содержания снимка. Наиболее значимый прирост качества нейросети демонстрируют на изображениях с антропогенными объектами, обладающими выраженной геометрией. Для классов buildings (здания), harbor (порт) и denseresidential (плотная застройка) преимущество модели rgb_rcan над бикубической интерполяцией достигает 1.0–1.3 дБ. Показательным примером служит класс denseresidential, где метрика возрастает с 31.92 дБ до 33.23 дБ. Это объясняется тем, что глубокие модели успешно обучаются восстанавливать прямые линии и резкие границы искусственных сооружений, которые классические методы неизбежно размывают.

Противоположная ситуация наблюдается при обработке природных ландшафтов. На снимках со сложными, хаотичными текстурами — forest (лес), beach (пляж) или river (река) — выигрыш от использования нейросетей оказывается минимальным и составляет всего 0.1–0.2 дБ. Так, для класса forest результаты rgb_rcan (31.34 дБ) практически не отличаются от бикубической интерполяции (31.24 дБ). Это свидетельствует о том, что восстановление стохастических деталей, таких как листва деревьев или рябь на воде, остается сложной задачей даже для глубоких сверточных сетей.

Отдельно стоит отметить группу классов с высокими абсолютными значениями метрик, таких как baseball diamond и golfcourse. Для них показатели превышают 38 дБ вне зависимости от используемого метода. Это связано с наличием на таких снимках обширных областей однородного цвета (газоны, грунтовые площадки), которые легко поддаются восстановлению любым алгоритмом, нивелируя разницу между сложными и простыми подходами.

Количественные метрики не всегда способны в полной мере отразить перцептивное качество изображения. Для верификации численных данных был проведен визуальный анализ восстановленных фрагментов на примерах, представляющих различные типы ландшафта.

3.4. Визуальный анализ результатов

Количественные метрики не всегда способны в полной мере отразить качество восстановления изображения, поэтому был проведен визуальный анализ восстановленных фрагментов. Для сравнения были отобраны три характерных примера, демонстрирующих работу алгоритмов с мелкими объектами, геометрическими структурами и природными текстурами.

На Рисунке 3.5 продемонстрирован результат обработки изображения из класса parkinglot, который является сложным из-за высокой плотности мелких объектов.

Рисунок 3.5 — Сравнение методов на примере плотной парковки

Отчетливо видно, что классические методы и y_fsrcnn не справляются с передачей высокочастотных деталей: ряды автомобилей сливаются в размытые полосы, а границы между отдельными машинами становятся неразличимыми. В то же время модели rgb_ressr и rgb_rcan демонстрируют качественный скачок: они успешно разделяют плотно стоящие автомобили, восстанавливая темные промежутки асфальта между ними и возвращая объектам их прямоугольную форму.

Далее был рассмотрен пример восстановления городской инфраструктуры с четкой геометрией из класса buildings (Рисунок 3.6).

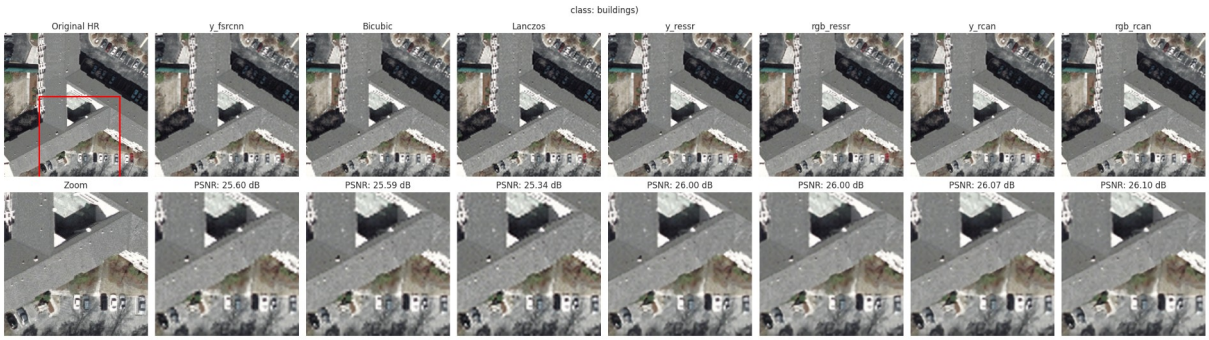


Рисунок 3.6 — Восстановление геометрии зданий и теней

На данном примере критерием качества является сохранение прямых линий и резкости границ теней. Бикубическая интерполяция приводит к размытию краев крыши и скруглению углов здания. Модель rgb_rcan показывает наилучший результат: границы теней становятся резкими, а геометрия здания восстанавливается без искажений. Это подтверждает выводы, сделанные ранее при анализе тепловой карты: нейросетевые методы эффективны при работе с антропогенными объектами, имеющими регулярную структуру.

Для оценки границ применимости методов также был проанализирован снимок из класса agricultural (Рисунок 3.7), содержащий природную текстуру посадок.

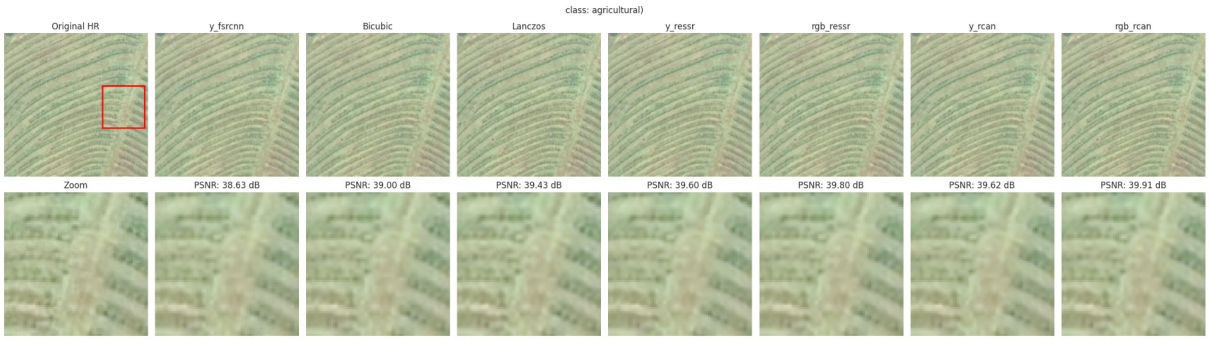


Рисунок 3.7 — Сравнение методов на примере растительности

Здесь разница между алгоритмами выражена наименее ярко. Нейросетевые модели не синтезируют новые детали листвы или почвы. Это объясняется стохастическим характером текстуры, восстановление которой является сложной задачей для моделей, обучаемых на минимизацию пиксельной ошибки. Визуаль-

ный анализ подтверждает, что основное преимущество глубоких нейросетей заключается в корректном восстановлении геометрии объектов, а не в генерации природных текстур.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы был проведен сравнительный анализ методов повышения разрешения спутниковых снимков, включающий реализацию и тестирование классических интерполяционных алгоритмов и нейросетевых архитектур различной глубины. Была разработана методика эксперимента, позволившая оценить эффективность подходов как по объективным метрикам, так и с точки зрения визуального восприятия и вычислительной сложности.

Результаты экспериментов показали, что классические методы обеспечивают высокую скорость обработки, но приводят к потере детализации. Применение простых нейросетевых архитектур также оказалось малоэффективным, показав качество на уровне интерполяции. Наилучшие результаты продемонстрировали глубокие остаточные сети, обученные на RGB-изображениях.

Сравнительный анализ выявил, что архитектура ResSR является наиболее сбалансированным решением для практического применения. Она не уступает по качеству, но обладает более высокой производительностью, доказав, что внедрение механизмов внимания в данной задаче не дает критического преимущества. Также установлены границы применимости методов: нейросети дают максимальный эффект на объектах с регулярной геометрией, в то время как на хаотичных природных текстурах их преимущество перед классическими алгоритмами минимально.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

[1] Dong C. et al. Image super-resolution using deep convolutional networks // IEEE transactions on pattern analysis and machine intelligence. – 2015. – V. 38. – №. 2. – C. 295-307.

[2] Dong C. et al. Accelerating the super-resolution convolutional neural network // European conference on computer vision. – 2016. – C. 391-407.

[3] Zhang Y. et al. Image super-resolution using very deep residual channel attention networks // Proceedings of the European conference on computer vision (ECCV). – 2018. – C. 286-301.