Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки: 01.03.02 Прикладная математика и информатика

РАБОТА ПРОВЕРЕНА
Рецензент, профессор кафедры ВМех,
к.ф.-м..н.
_____/ М.В. Плеханова
«____»_____20___г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
профессор
_____/А.А.Замышляева
«____»_____20___ г.

Сверхразрешение изображений с помощью генеративно-состязательной нейронной сети

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.03.02.2020.057.ПЗ ВКР

Консультант, руководитель
Научно-исследовательской
Группы челябинского офиса
ООО «Цифровая собственность»
_____/Р.Р. Абясов
«____»_____2020 г.

Руководитель работы,
к.ф.-м.н., доцент
_____/Т.В. Карпета
«____»_____2020 г.

Автор работы
Студент группы ЕТ-412
_____/ М.А. Васюк
«____»_____2020 г.

Нормоконтролер,
ст. преподаватель
_____/Н.С. Мидоночева
«____»_____2020 г.

Челябинск
2020

АННОТАЦИЯ

Васюк М.А. Сверхразрешение изображений с помощью генеративно-состязательной сети. – Челябинск: ЮУрГУ, ЕТ-412, 74 с., 37 ил., 8 табл., библиогр. список – 30 наим.

Целью данной работы является разработка искусственной нейронной сети для повышения разрешения изображений без потери качества. В данной работе решается задача сверхразрешения изображений с использованием генеративно-состязательной нейронной сети.

Данное решение реализовано на языке программирования Python с использованием библиотеки глубокого обучения PyTorch. Для подготовки и обработки данных использовались библиотека для работы с изображениями OpenCV и библиотека для работы с многомерными массивами NumPy. По окончанию обучения был проведен сравнительный анализ качества работы нейронной сети с работой других решений поставленной задачи.

Реализованная модель может применяться в обработке фото с целью повышения визуального качества.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современное время качество цифровой фотографии с каждым годом становится всё лучше и лучше. Разрешением фотографии называется величина, которая равна количеству всех пикселей изображения, деленное на единицу площади. Чем выше разрешение изображения, тем больше деталей на изображении можно различить, а, соответственно, из такого изображения можно извлечь больше полезной информации.

В большинстве задач, решаемых с помощью искусственных нейронных сетей, подразумевается, что в качестве данных для обучения используется набор изображений в высоком разрешении. Примером может послужить задача классификации изображений, при решении которой алгоритмы компьютерного зрения будут работать гораздо лучше, получая на вход изображения в высоком разрешении. Таким образом, одним из вариантов улучшения работы нейронной сети может быть предобработка набора данных для обучения путем повышения разрешения изображений, из которых он состоит.

С одной стороны, проблему можно решить аппаратным путем, то есть отснять фотографии, используя фотоаппарат, обладающий более высокой разрешающей способностью, но стоит учесть, что такой процесс может оказаться дорогостоящим и трудоемким. С другой стороны, когда уже имеется какой-то набор фотографий, может возникнуть ситуация, в которой использование такого метода в принципе невозможно. В таком случае разумно прибегнуть к программным методам решения, например, к нейросетевым методам, позволяющим добиться лучшего качества среди существующих решений. Подобные алгоритмы повышают разрешение изображения за счет дополнительной информации, которую они получают во время процесса обучения на большом количестве изображений в высоком разрешении либо получают, используя информацию о соседних пикселях.

Благодаря такому подходу к подготовке данных перед подачей на вход сети, можно заметно улучшить эффективность работы многих алгоритмов компьютерного зрения, повысив разрешение изображений, используемых для обучения этих алгоритмов.

Целью данной работы является разработка искусственной нейронной сети для повышения разрешения изображений без потери качества.

Для достижения данной цели необходимо решить следующие задачи:

1) исследование существующих методов для решения задачи повышения разрешения изображения без потери качества;

2) сбор и подготовка данных для обучения нейронной сети;

3) разработка математической модели искусственной нейронной сети;

4) обучение сети и оценка качества модели на экспериментальных данных.

1 ПОВЫШЕНИЕ РАЗРЕШЕНИЯ ИЗОБРАЖЕНИЙ БЕЗ ПОТЕРИ КАЧЕСТВА

1.1 Методы повышения разрешения изображений

1.1.1 Интерполяционные методы

Интерполяция – операция приближения функции, которая задана внутри некоторого определенного интервала в отдельных точках [8]. Рассмотрим суть задачи интерполяции в одномерном случае. Пусть на отрезке $[a, b]$ заданы $n + 1$ точек x_i $(i = 0, 1, 2, \dots, n)$. Такие точки называются узлами интерполяции. Также в этих точках известны значения некоторой функции $f(x)$: $f(x_0), f(x_1), \dots, f(x_n)$. Необходимо для этих точек восстановить такую функцию $F(x)$ (интерполирующую функцию), которая принимает в заданных точках отрезка те же значения, что и функция $f(x)$, т.е. $F(x_0) = f(x_0)$, $F(x_1) = f(x_1), \dots, F(x_n) = f(x_n)$.

Интерполяционную формулу $y = F(x)$ получают различными способами. Найденная функция используется для вычисления значений функции $f(x)$ в тех точках отрезка $[a, b]$, где ее значения изначально не были известны. Такая операция называется интерполяцией функции $f(x)$.

Дискретную модель цифрового изображения размерами $m \times n$ можно представить в виде двумерной матрицы:

$$f(x, y) = \begin{pmatrix} f(0,0) & f(0,1) & \dots & f(0,n-1) \\ \vdots & \vdots & \dots & \vdots \\ f(m-1,0) & f(m-1,1) & \dots & f(m-1,n-1) \end{pmatrix}. \quad (1.1)$$

Элемент данной матрицы называется пикселем и является некоторым вещественным числом. Таким образом, изображение есть дискретно заданная функция двух переменных, т. е. мы имеем множество точек $\{f(x_i, y_j),\ i = \overline{0,n},\ j = \overline{0,m}\}$.

Для решения таких задач, как повышение разрешения изображения, используется интерполяция. Интерполяция изображений основывается на

значениях соседних пикселей и вычисляет подходящие приближения в цвете и яркости пикселя [24].

Рассмотрим наиболее часто используемые методы интерполяции изображений.

1. Метод ближайшего соседа. Суть данного метода заключается в создании нового пикселя, основываясь на информации, полученной от сосед-них относительно него пикселей. Этот метод хоть и не точный, но является достаточно быстрым. Такой метод повышения разрешения приводит к увеличению восприятия растровой структуры пикселей, которая при снижает визуальное качество увеличенных изображений. Для большинства изображений данный метод не подходит, поскольку повышает вероятность возникновения на изображениях «зубчатых» краев. Помимо этого, используя метод ближайшего соседа, часть информации может быть утеряна, а часть – продублирована.

$$f(x,y) = f\left(x \cdot \frac{W_{old}}{W_{new}}, y \cdot \frac{H_{old}}{H_{new}}\right), \tag{1.2}$$

где W_{old}, H_{old} – ширина и высота изначального изображения;

W_{new}, H_{new} – ширина и высота конечного изображения.

2. Билинейная интерполяция является расширением линейной интерполяции для функции двух переменных. Основная идея состоит в том, чтобы произвести линейную интерполяцию сначала вдоль одной оси коор-динат, затем вдоль другой. Используя данную модификацию, изображения, интерполированные билинейным методом, выглядят значительно более четко, чем результат работы метода ближайшего соседа.

Рассмотрим принцип метода билинейной интерполяции на примере. Пусть требуется интерполировать значение некоторой функции f в точке $P(x,y)$. Изначально известны значения данной функции в окружающих P точках $Q_{11} = (x_1, y_1), Q_{12} = (x_1, y_2), Q_{21} = (x_2, y_1), Q_{22} = (x_2, y_2)$ (рисунок 1.1).

На первом шаге производится линейная интерполяция значений вспомогательных точек $R_1 = (x_1, y_1)$ и $R_2 = (x_2, y_2)$ вдоль оси абсцисс:

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}), \quad (1.3)$$

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}). \quad (1.4)$$

Далее проводится интерполяция между дополнительными точками R_1 и R_2 вдоль оси ординат:

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2). \quad (1.5)$$

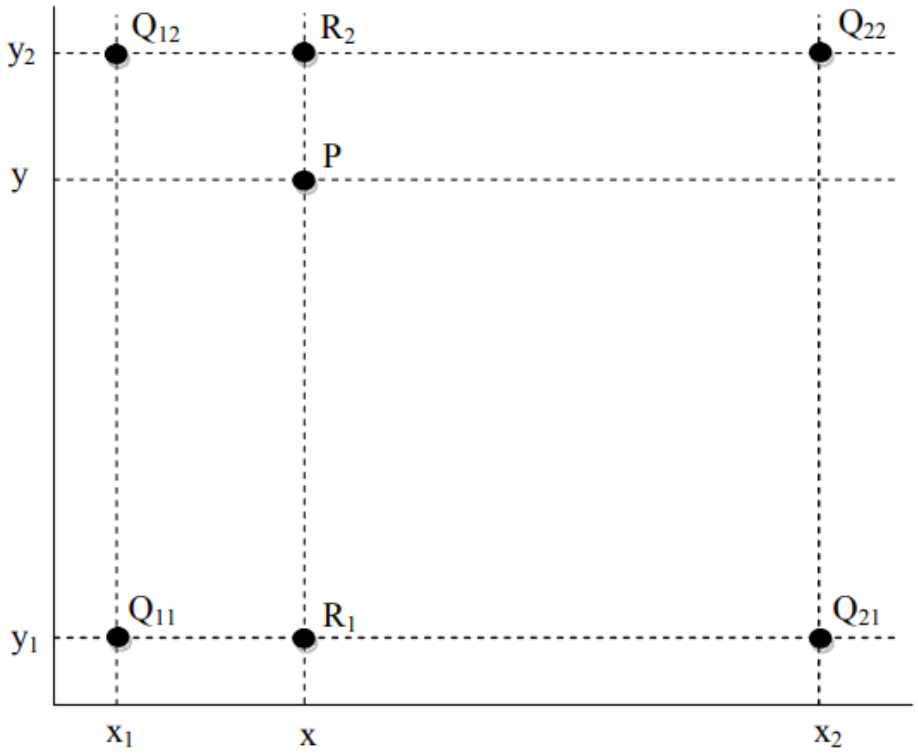


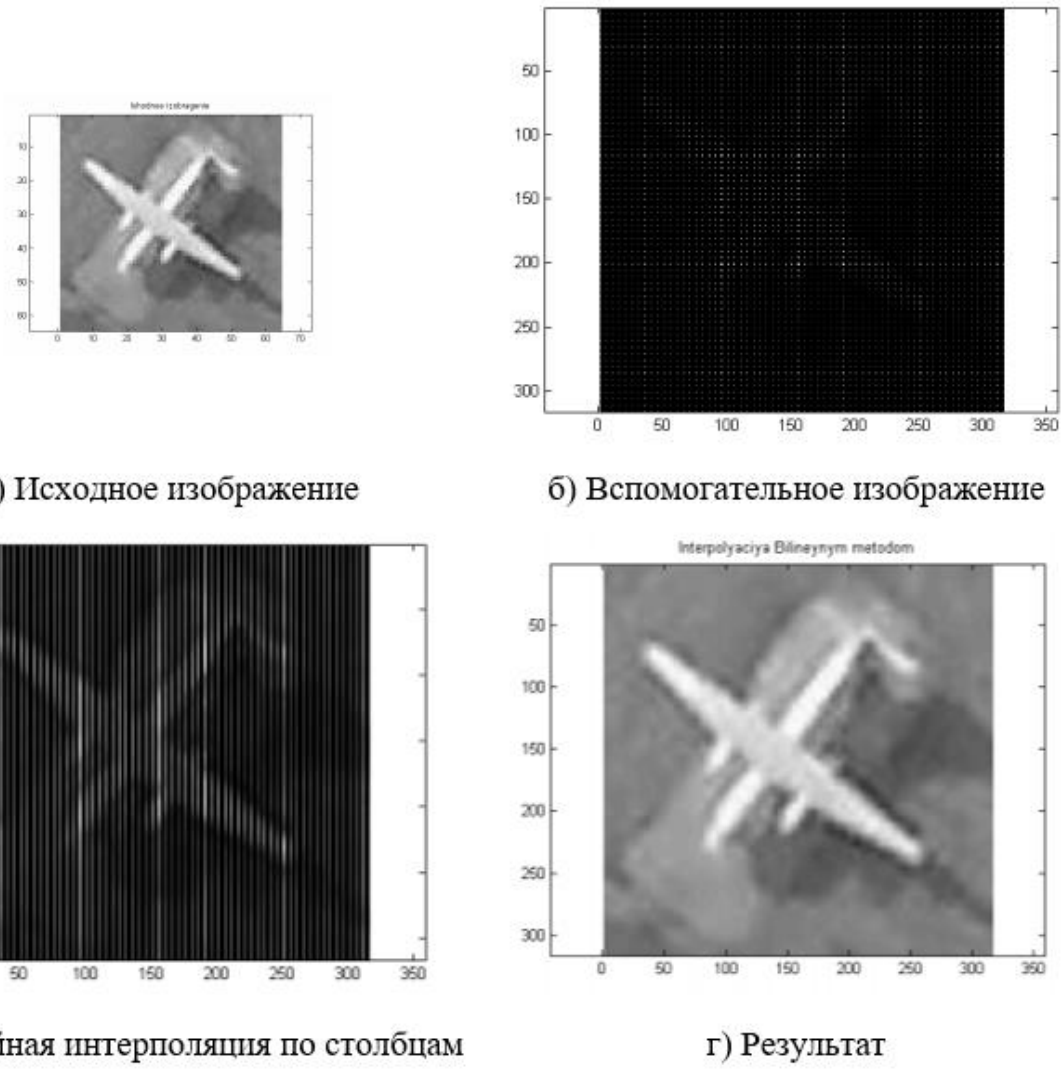
Рисунок 1.1 – Билинейная интерполяция в точке P

Принцип получения изображения, интерполированным билинейным методом, представлен на рисунке 1.2.

На первом этапе формируется интерполированное линейно по строкам изображение. Расстояние, на которое разносятся точки вспомогательного изображения между собой, зависит от выбранного коэффициента масштабирования (рисунок 1.2, б). Затем производится линейная интерполяция по столбцам (рисунок 1.2, в). На последнем этапе формируется

результирующее интерполированное билинейным методом изображение (рисунок 1.2, г) [11].

3. Бикубический метод интерполяции использует значения восьми прилегающих пикселей и дает относительно наилучшие результаты, поскольку за счет повышения контрастности переходов возникает эффект повышения резкости изображения [11].



а) Исходное изображение

б) Вспомогательное изображение

в) Линейная интерполяция по столбцам

г) Результат

Рисунок 1.2 – Билинейная интерполяция (увеличение в 5 раз)

Бикубическая интерполяция является модификацией кубической интерполяции в случае функции двух переменных.

Рассматривается два подхода к реализации бикубической интерполяции: бикубическая интерполяция сплайнами и последовательная кубическая интерполяция.

При бикубической интерполяции сплайнами для интерполяции значения функции $f(x,y)$ в точке $P(x,y)$, которая лежит внутри квадрата $[0,1]\times[0,1]$, при известном значении функции f в шестнадцати соседних точках $(i,j), i=-1,\dots,2,\ j=-1,\dots,2$ общий вид функции задающей интерполированную поверхность, может быть записан следующим образом:

$$p(x,y)=\sum_{i=0}^{3}\sum_{j=0}^{3}a_{ij}x^i y^j. \tag{1.6}$$

Для того, чтобы найти коэффициенты a_{ij}, необходимо подставить в формулу 1.6 значения функции в известных шестнадцати точках.

Единожды найденные коэффициенты a_{ij} можно использовать для многократного вычисления интерполированного значения функции в произвольных точках квадрата $[0,1]\times[0,1]$.

Последовательная интерполяция заключается в том, что для нахождения интерполированного значения можно сначала произвести кубическую интерполяцию в одном направлении, а затем в другом.

Для функции $f(x)$ с известными значениями $f(-1), f(0), f(1), f(2)$ может быть построен кубический сплайн: $p(x)=\sum_{i=0}^{3}b_i x^i$, или в матричном виде:

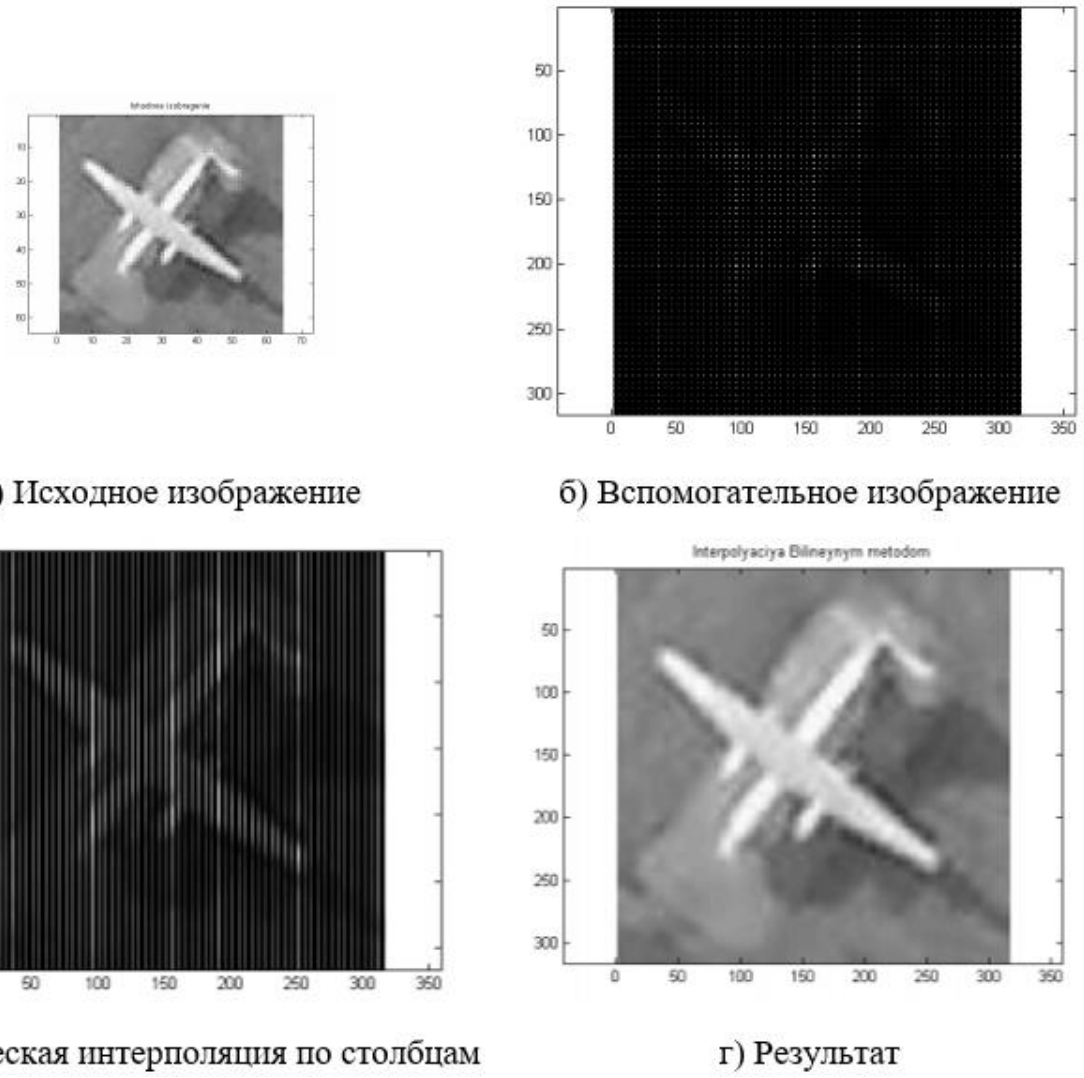
$$p(x)=\begin{pmatrix}1 & x & x^2 & x^3\end{pmatrix}\begin{pmatrix}b_0\\ b_1\\ b_2\\ b_3\end{pmatrix}, \tag{1.7}$$

где $\begin{pmatrix}b_0\\ b_1\\ b_2\\ b_3\end{pmatrix}=A\begin{pmatrix}f(-1)\\ f(0)\\ f(1)\\ f(2)\end{pmatrix};\ A=\frac{1}{6}\begin{pmatrix}0 & 6 & 0 & 0\\ -2 & -3 & 6 & 1\\ 3 & -6 & 3 & 0\\ -1 & 3 & -3 & 1\end{pmatrix}$.

Таким образом, для нахождения интерполированного значения $p(x,y)$ в квадрате $[0,1]\times[0,1]$ можно сначала рассчитать четыре промежуточных значения $p(x,-1),\ p(x,0),\ p(x,1),\ p(x,2)$ для зафиксированного x, затем через полученные четыре точки построить кубический сплайн, и этим завершить вычисление $p(x,y)$.

$$p(x,y) = \begin{pmatrix} 1 \\ y \\ y^2 \\ y^3 \end{pmatrix}^T A \begin{pmatrix} f(-1,-1) & f(0,-1) & f(1,-1) & f(2,-1) \\ f(-1,0) & f(0,0) & f(1,0) & f(2,0) \\ f(-1,1) & f(0,1) & f(1,1) & f(2,1) \\ f(-1,2) & f(0,2) & f(1,2) & f(2,2) \end{pmatrix} A^T \begin{pmatrix} 1 \\ x \\ x^2 \\ x^3 \end{pmatrix}. \quad (1.8)$$

Принцип получения интерполированного изображения методом бикубической интерполяции аналогичен принципу при использовании билинейной интерполяции (рисунок 1.3).



а) Исходное изображение

б) Вспомогательное изображение

в) Кубическая интерполяция по столбцам

г) Результат

Рисунок 1.3 – Бикубическая интерполяция (увеличение в 5 раз)

Любой интерполяционной метод в общем случае можно представить, как операцию линейной свертки:

$$f(x) = \sum_{i=-\infty}^{+\infty} F(i)K(i-x), \quad (1.9)$$

где $F(i)$ – интенсивность (яркость) i-го пикселя изображения;

K – весовая функция или ядро свертки.

В двумерном случае:

$$f(x) = \sum_{i,j=-\infty}^{+\infty} F(i,j)K(i-x)K(j-y). \quad (1.10)$$

Свертка – базовая операция в задачах цифровой обработки сигналов.

Пусть даны два дискретных сигнала $a(n),\ n = 0, \dots, N-1$ и $b(n), n = 0, \dots, M-1$. Линейной сверткой двух сигналов $a(n)$ и $b(n)$ называется дискретный сигнал следующего вида:

$$s(n) = \sum_{m=0}^{n} a(m)b(n-m), n = 0, \dots, (N+M-2). \quad (1.11)$$

Для вычисления $s(n)$ сигналы $a(n)$ и $b(n)$ сдвигают друг относительно друга, перемножают соответствующие элементы и складывают их. Функция K задает вес пикселя и определяет отношение удаленности пикселей между собой. Таким образом, чем дальше пиксель находится от интерполированного значения пикселя (x, y), тем меньше его вес.

При этом предполагается, что веса вне окрестности нулевые, т.е. $a(n) = 0$, при $n < 0$ и $n \geq N$, и $b(n) = 0$, при $n < 0$ и $n \geq M$ [25].

Качество результирующего изображения при обработке одним из интерполяционных методов в таком виде зависит от выбора ядра (весовой функции) K, с помощью которого выполняется вышеупомянутая операция свертки (рис. 1.4–1.7).

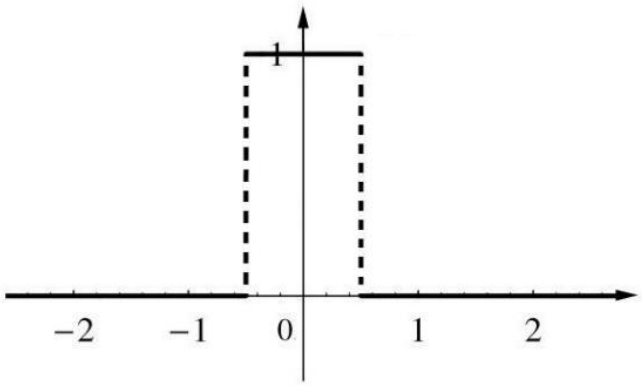


Рисунок 1.4 – Ядро одномерной интерполяции по соседним элементам

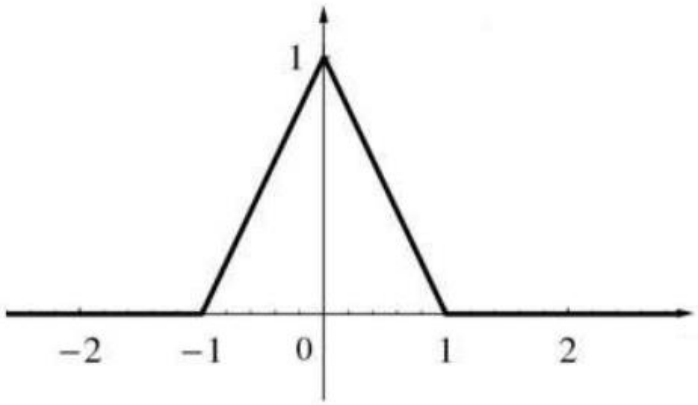


Рисунок 1.5 – Ядро линейной интерполяции

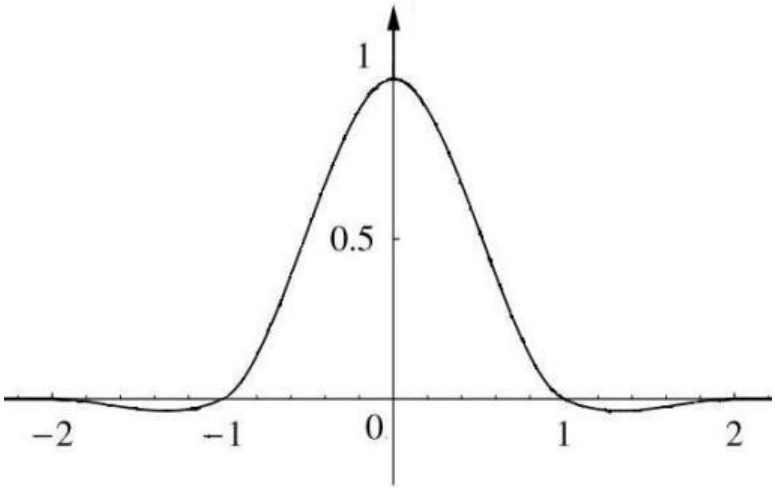


Рисунок 1.6 – Ядро бикубической интерполяции

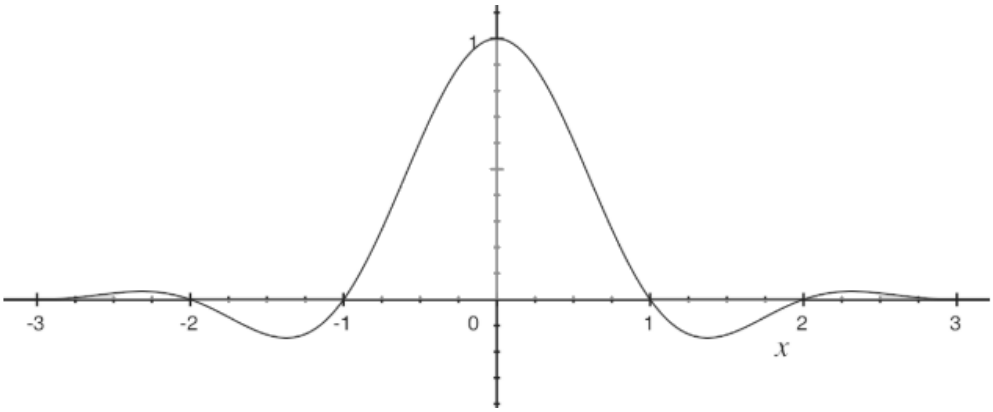


Рисунок 1.7 – Ядро интерполяции Ланцоша

Результаты, полученные при помощи интерполяционных методов сверхразрешения, могут иметь ряд существенных недостатков: искажения геометрии мелких текстур и деталей, возникновение ложных узоров и др.

Использование интерполяции для повышения разрешения изображений в большинстве случаев приводит к возникновению трех негативных эффектов (рисунок 1.8).

1. Алиасинг (эффект «ступенчатости» линий). Связан с проблемами повышения разрешения линий, не являющихся параллельными какой-либо из осей координат. Алиасинг возникает, когда происходит пересечение таких линий со строками или столбцами матрицы пикселей под небольшим углом. Часть линии может находиться в одном ряду матрицы, а часть – в другом. Отсюда, может возникнуть неопределенность: в каком ряду располагать пиксель или вовсе располагать его в обоих рядах.

2. Размытие изображений. Данный негативный эффект возникает там, где находятся объекты с выраженными, четкими границами. Его появление связано с тем, что при повышении разрешения изображений граница объекта становится размазанной и не четкой.

3. Эффект Гиббса. В местах резких перепадов интенсивности яркости соседних пикселей на изображениях могут проявляться ореолы (свечение). При большом коэффициенте повышения разрешения данный эффект становится слишком заметным для человеческого глаза, что делает его сопоставимым с потерей цветности [24].

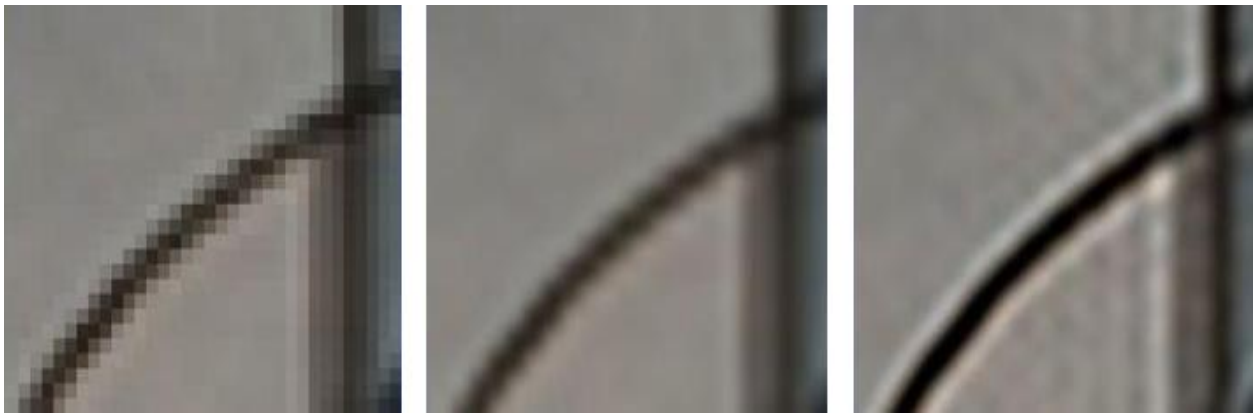


Рисунок 1.8 – Артефакты, возникающие при использовании интерполяции

Таким образом, интерполяционные методы ограничиваются применением для простых и не детализированных изображений (например, текстур), которые не содержат контрастных элементов. Для таких изображений количество возникающих артефактов будет сведено к минимуму.

1.1.2 Нейросетевые методы

В области программной обработки изображений существует семейство методов сверхразрешения. Такие методы позволяют повысить изначальное разрешение изображения, преодолев при этом фактическое разрешение цифровой камеры, записавшей изображение, и оптический предел ее объектива.

В отличие от традиционных интерполяционных методов повышения разрешения изображений, цель сверхразрешения немного иная: методы сверхразрешения, на самом деле, направлены на оценку недостающих деталей высокого разрешения, которых нет в исходном изображении, путем добавления новых наиболее вероятных.

Для достижения данной цели наиболее часто в последнее время изучают два подхода сверхразрешения для:

1) многокадровый – метод, основанный на использовании нескольких изображений одного объекта;

2) однокадровый – метод, обучающийся на некоторой базе образцов.

Многокадровый метод сверхразрешения, как следует из названия, полагается на наличие нескольких кадров, взаимно смещенных и, возможно, геометрически преобразованных. Эти несколько кадров объединяют особым способом для получения единого изображения в высоком разрешении. Таким образом, выходное изображение содержит информацию, полученную из входных изображений.

Однокадровый метод сверхразрешения, соответственно, имеет в наличии лишь один кадр низкого разрешения, на основе которого должно получиться изображение в высоком разрешении, т. е. недостающая информация будет создана «с нуля» [24].

Под однокадровым методом подразумевается нейронная сеть, обучаемая на большом наборе данных, из которого сеть находит скрытые зависимости и изучает взаимное расположение пикселей друг относительно друга. Таким образом обученная сеть, получившая на вход изображение, разрешение

которого нужно увеличить, будет пытаться «додумывать» недостающие пиксели на основе тех образцов, на которых она обучалась. Отсюда следует, что чем больше будет разнообразных образцов в обучающей выборке, тем визуально лучше и реалистичнее будет отрабатывать нейронная сеть [29].

1. Сверточная нейронная сеть – одна из разновидностей нейронных сетей, предназначенных для активного анализа преимущественно двухмерных и трехмерных данных. Архитектура сверточной нейронной сети была впервые предложена французским ученым Яном Лекуном в конце 80-х годов прошлого века. Свое название сеть получила за счет использующейся в ней операции свертки [7].

Свертка – операция над парой матриц A (размера $n \times n$) и B (размера $m \times m$), результатом которой является матрица $C = A \cdot B$ размера $(n - m + 1) \times (n - m + 1)$ [14]. Каждый элемент результат вычисляется как скалярное произведение матрицы B и некоторой подматрицы A такого же размера (подматрица определяется положением элемента в результате):

$$C_{i,j} = \sum_{u=0}^{m-1} \sum_{v=0}^{m-1} A_{i+u,j+v} B_{u,v}. \quad (1.12)$$

На рисунке 1.9 можно увидеть, как матрица B «скользит» по матрице A, и в каждом положении вычисляется скалярное произведение матрицы B и соответствующей части матрицы A, на которую она сейчас наложена. Получившееся число записывается в соответствующий элемент результата. В качестве матрицы A, как правило, выступает изображение, а матрицу B называют фильтром или ядром свертки [9, 18].

Использование сверточной нейронной сети для решения задачи повышения разрешения, как правило, подразумевает несколько этапов. На первом этапе изображение низкого разрешения проходит предварительную обработку, например, с помощью одного из интерполяционных методов его увеличивают до разрешения выходного изображения, тем самым получая начальное приближение изображения высокого разрешения.

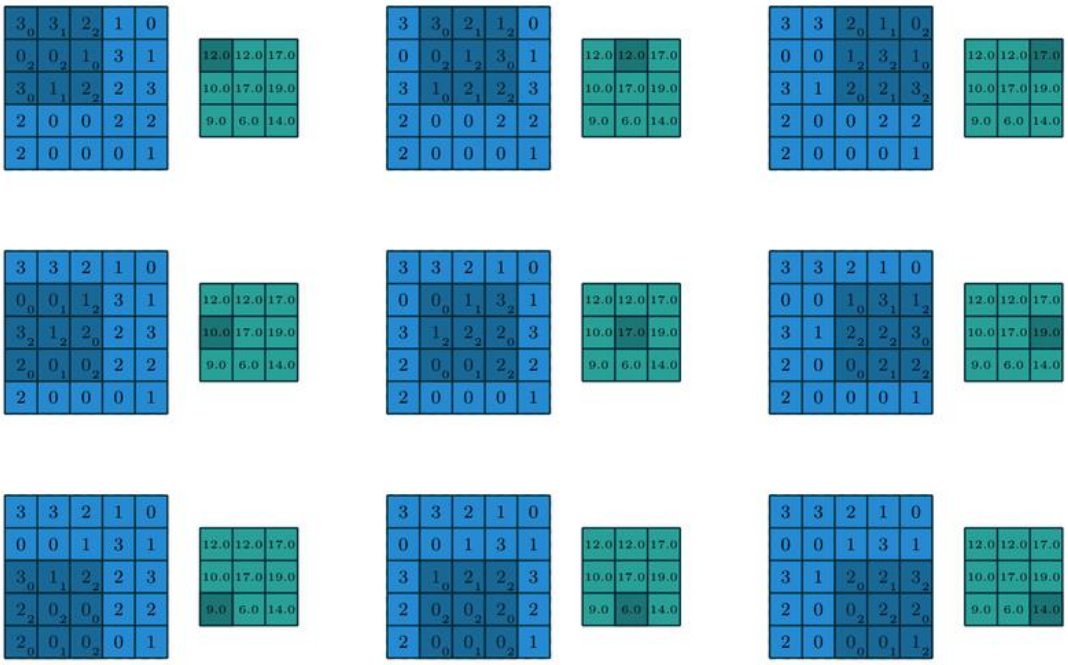


Рисунок 1.9 – Выполнение операции свертки

Затем интерполированное изображение подается на вход сверточной сети. Этап предварительной обработки можно поместить в первый слой сверточной сети, представив операцию интерполяции в качестве сверточного слоя с соответствующим ядром. При этом стоит учесть, что данный слой должен быть исключен из цикла обучения сети. Сама сверточная сеть выполняет три основных операции (рисунок 1.10).

1. Извлечение признаков изображения. Данная операция извлекает из изображения низкого разрешения (или интерполированного изображения) карты признаков и представляет каждый такой признак как вектор в многомерном пространстве. Извлечение признаков происходит с помощью применения ко входному изображению нескольких блоков, состоящих из сверточных и активационных слоев. Таким образом, мы извлекаем из изображения основную информацию, которая в дальнейшем будет использоваться для восстановление недостающих пикселей при повышении разрешения.

2. Нелинейное преобразование. Данная операция нелинейным образом преобразует многомерные векторы, полученные на предыдущем шаге, в другие многомерные векторы. Нелинейное преобразование выполняется с помо-

щью одного или нескольких сверточных слоев, за каждым из которых следует слой активации.

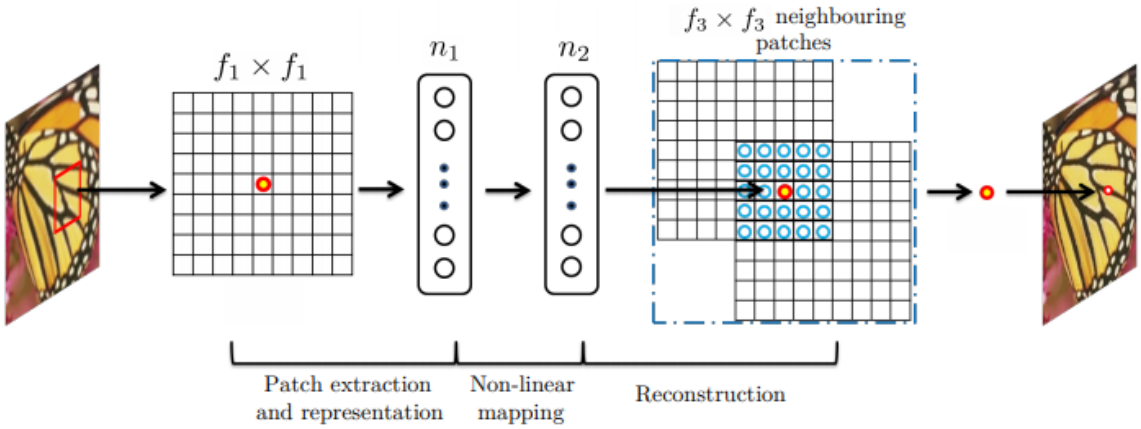


Рисунок 1.10 – Поэтапное представление сверточной нейронной сети

3. Реконструкция. Данная операция объединяет все нелинейно преобразованные многомерные векторы с целью дальнейшего получения окончательного изображения высокого разрешения. В сверточной сети операция реконструкции представлена в виде блоков, состоящих из деконволюционных и активационных слоев [19].

Деконволюция (десвертка или транспонированная свертка) – операция обратной свертки, целью которой является воссоздание сигнала, существовавшего до того, как произошла свертка (рисунок 1.11).

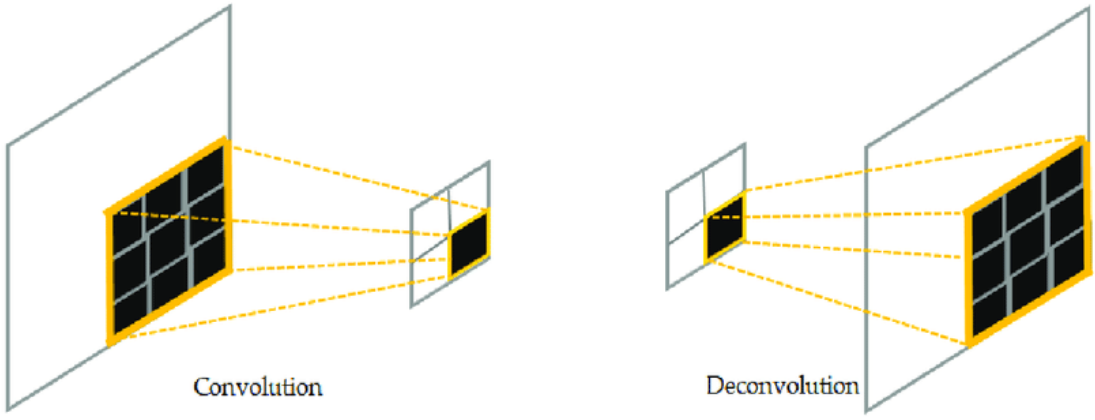


Рисунок 1.11 – Операции свертки и обратной свертки

Обычно использование таких сетей имеет смысл, если известны характеристики свертки, с помощью которой было преобразовано изображение. Однако, так как мы имеем дело с изображениями в низком разре-

шении, к которым в принципе не применялась операция свертки, то деконволюция выполняется «вслепую» либо предполагается, что свертка выполняла роль усредняющего фильтра [13]. Тогда операция деконволюции будет преобразовывать один пиксель в несколько близких ему по интенсивности пикселей.

2. Генеративно-состязательная сеть (Generative adversarial network, GAN) – архитектура, состоящая из двух подсетей (дискриминатора и генератора), которые настроены на противодействие друг с другом. Отсюда нейронная сеть и получила свое название – генеративно-состязательная. Такой тип нейросетевых архитектур является одним из наиболее современных и эффективных методов обучения без учителя (unsupervised learning) [12].

Обучение без учителя (unsupervised learning) – один из разделов машинного обучения, изучающий класс задач, связанных с обработкой данных, в которых изначально известны только описания объектов множества (обучающей выборки), и необходимо обнаружить внутренние зависимости, закономерности, существующие между объектами [3, 5].

Обучению без учителя противопоставляют обучение с учителем, в котором для каждого объекта обучающего набора известен «эталонный ответ», и необходимо восстановить зависимость между объектами и ответами.

Генеративно-состязательные нейронные сети впервые были предложены в работе 2014-го года исследователем университета Монреаля – Яном Гудфеллоу. Саму идею состязательной тренировки сети многие называют самой интересной идеей в области машинного обучения за последнее десятилетие. Потенциал генеративно-состязательных сетей огромен, поскольку они могут имитировать любое распределение данных. Такие сети обучаются создавать структуры и образцы, максимально похожие на те, что существуют в реальном мире – реальные данные из области изображений, речи, музыки и т. д [5, 17].

Основная идея генеративно-состязательной сети заключается в обучении генератора и дискриминатора, которые постоянно соревнуются друг с другом. Одной из самых популярных аналогий является конкуренция между

банкиром и фальшивомонетчиком. Фальшивомонетчик, выступающий в качестве генерирующей сети, пытается воссоздать поддельные купюры, которые ничем не должны отличаться от настоящих. Банкир же, выступающий в роли дискриминатора, хочет научиться отличать друг от друга настоящие деньги и поддельные. В самом начале отличить реальные купюры от поддельных не составит большого труда, но в процессе обучения генерирующая сеть учится делать поддельные купюры всё лучше и лучше. В результате обучения, генератору необходимо научиться создавать поддельные купюры настолько хорошо, чтобы дискриминатор не сумел отличить их от реальных купюр [15].

Стоит отметить, что генератор не получает объекты реальных данных, на вход генератору поступает только зашумленное изображение или случайный вектор. Таким образом, чтобы обучать генератор, ему необходимо взаимодействие с дискриминатором. Дискриминатор, в свою очередь, получает на вход сгенерированные данные и объекты из реальной обучающей выборки. Ошибка обучения для дискриминатора вычисляется на основе того, откуда пришли данные. В процессе обучения генерирующая сеть обучается распределению исходной обучающей выборки и пытается воссоздать данные все более близкие к реальным. А дискриминатор обучается стать все более точным, чтобы отличить сгенерированный образец от оригинального (рисунок 1.12) [15, 16].

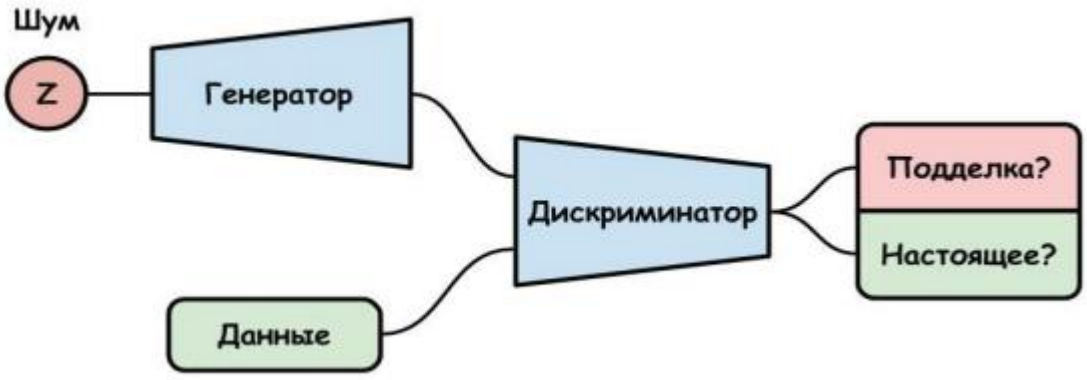


Рисунок 1.12 – Общая архитектура генеративно-состязательной сети

Одной из наиболее мощных и современных генеративно-состязательных сетей для решения задачи повышения разрешения изображений без потери

качества является сеть SRGAN [1,30]. На сегодняшний день это первое решение, способное восстанавливать исходные текстуры из изображений, уменьшенных в четыре раза. Средняя экспертная оценка показывает значительный прирост качества при использовании SRGAN: результаты оказались близки к значениям оригинальных изображений с высоким разрешением, чего не удавалось достичь с помощью других современных методов (рисунок 1.13).

Задача авторов работы состояла в том, чтобы обучить генеративную функцию, которая для входного изображения низкого разрешения оценивает соответствующее изображение высокого разрешения. Для этого мы обучаем генеративно-состязательную сеть как сверточную нейронную сеть прямого распространения с оптимизацией функции потерь восприятия (perceptual loss function).

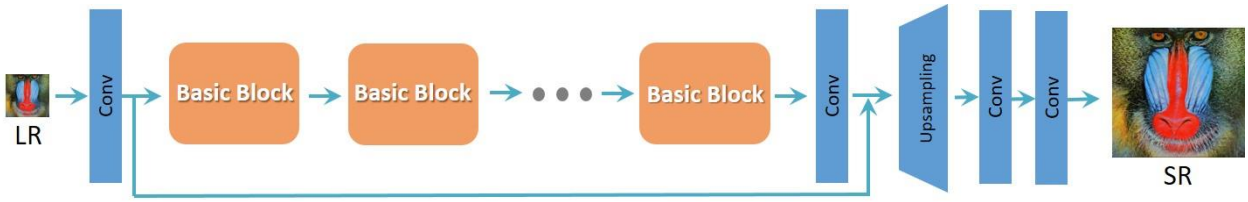


Рисунок 1.13 – Архитектура генератора сети SRGAN

В качестве дискриминатора также выступает сверточная нейронная сеть, которая вместе с генератором оптимизирует функцию потерь восприятия.

Главным преимуществом такого метода повышения является способность генеративно-состязательных сетей создавать изображения наиболее приближенные к тем, из которых состоит обучающий набор данных. Таким образом, обучая генерирующую сеть на реальных данных, мы можем добиться наилучшего качества по сравнению с традиционными и нейросетевыми методами повышения разрешения изображений.

1.2 Сервисы повышения разрешения изображений

1.2.1 Онлайн-сервис bigjpg.com

Bigjpg.com – это веб-приложение, которое с помощью нейронных сетей позволяет увеличить разрешение изображения (рисунок 1.14).

Данный сервис предоставляет возможность выбора степени увеличения разрешения (2, 4, 8 или 16) с возможностью выбора уровня подавления зашумленности в полученном изображении (рисунок 1.15).

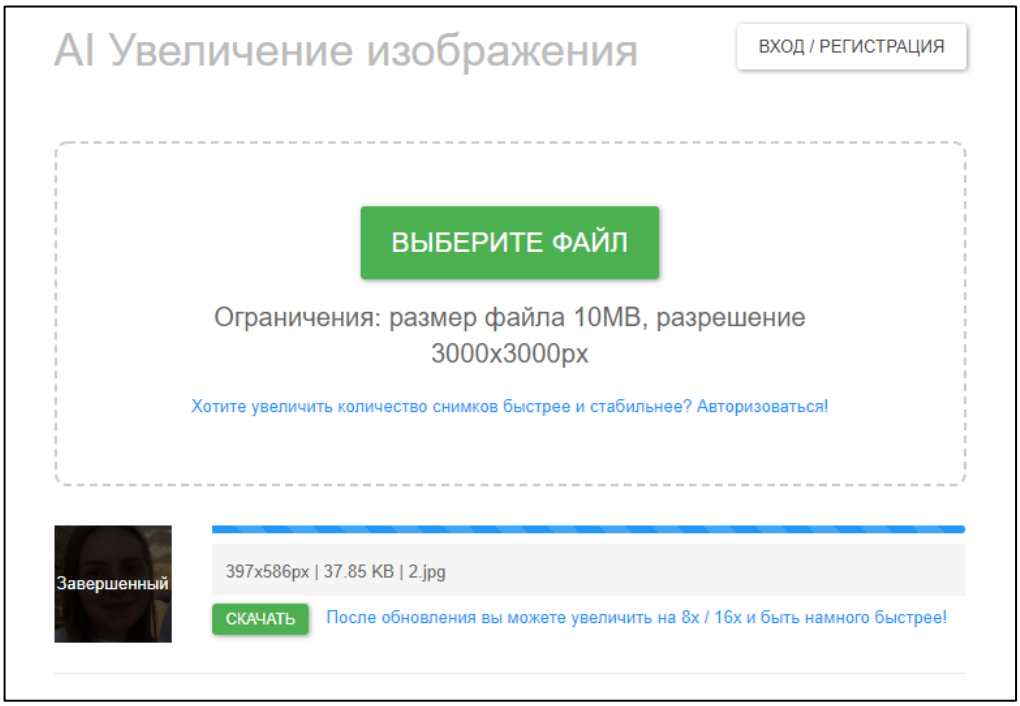


Рисунок 1.14 – Интерфейс сервиса bigjpg.com

Преимуществом данного сервиса является неплохая работа самой нейронной сети, также дополнительные возможности, с помощью которых можно управлять результатом генерации – выбор типа обрабатываемого изображения и устранение шума в выходном изображении.

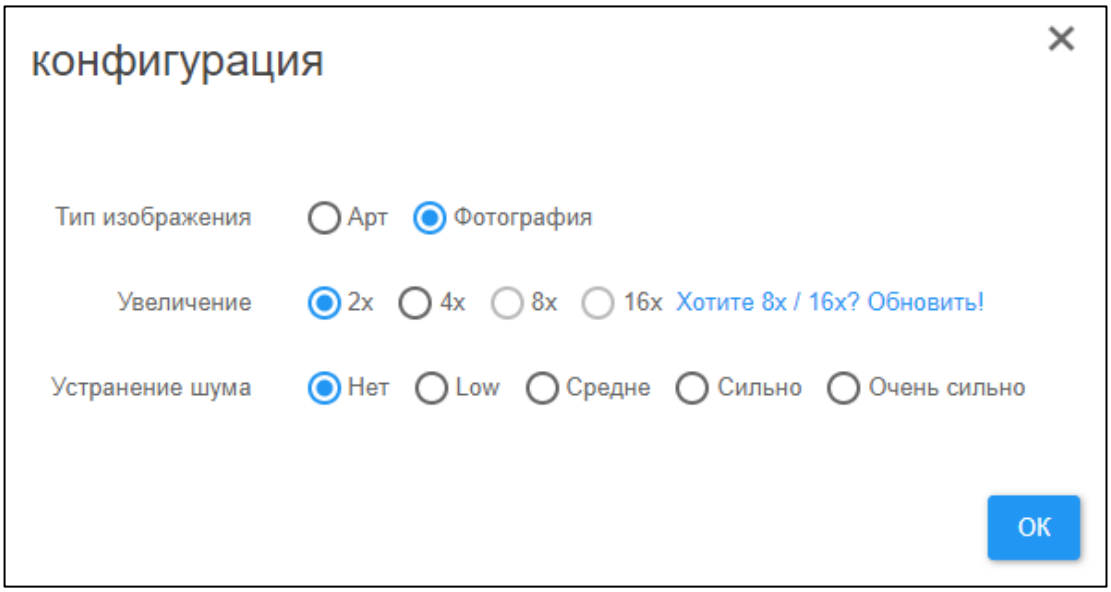


Рисунок 1.15 – Настройки повышения разрешения

Из недостатков приложения можно выделить следующие моменты: ограничение использования сервиса до 5 изображений в день, отсутствие бесплатного использования режимов увеличения в 8 и 16 раз, относительно долгая обработка изображения.

1.2.2 Онлайн-сервис Let’s Enhance

Let’s Enhance – веб-приложение, созданное украинскими программистами в 2017 году в целях увеличения изображения без потери качества с помощью искусственного интеллекта (рисунок 1.16).

Рисунок 1.16 – Онлайн-сервис увеличения разрешения Let’s Enhance

Также, как и у предыдущего решения, возможность 8- и 16-кратного увеличения становится доступной после приобретения платной подписки на данный сервис. Имеются ограничения, которые не позволяют воспользоваться сервисом более 5 раз в день. Еще одним из существенным недостатком сервиса является нестабильная работа: в 30% случаев система просто отказывается обрабатывать загруженное изображение и предлагает загрузить его заново.

Главные преимущества данной системы – это относительно быстрая скорость работы (даже без учета того, что изображение должно обработаться тремя нейронными сетями) и градация повышения разрешения, которая позволяет пользователю выбрать именно ту степень увеличения, которая ему требуется.

1.2.3 Онлайн-сервис pichance.com

В январе 2019-го года индийские разработчики представили новый веб-сервис pichance.com. Он использует искусственный интеллект, чтобы увеличивать изображения в 4 раза, при этом сохраняя исходную детализацию (рисунок 1.17).

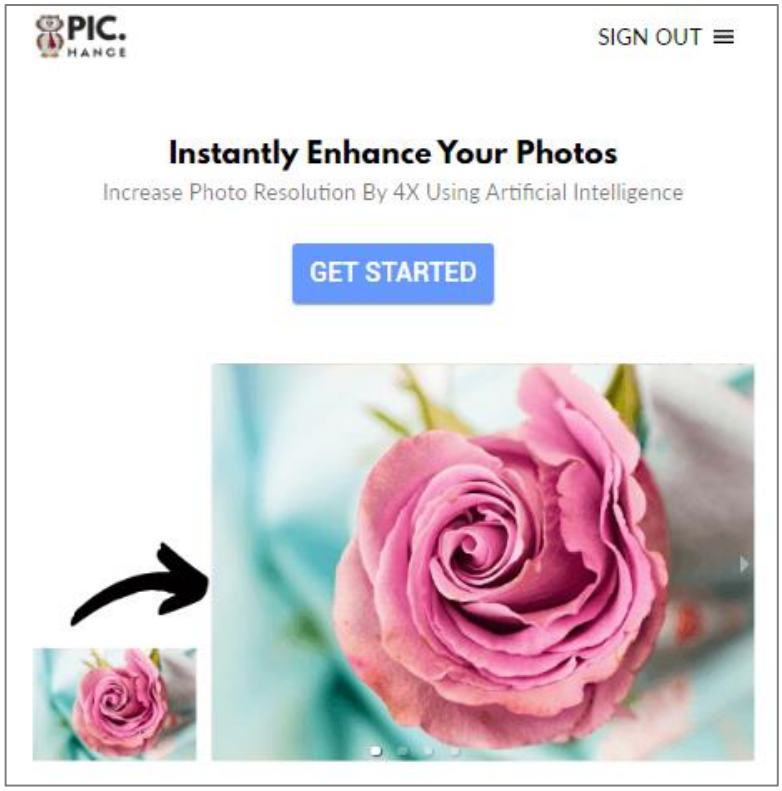


Рисунок 1.17 – Онлайн-сервис pichance.com

1.2.4 Онлайн-сервис Waifu2x

Сервис Waifu2x позволяет увеличивать в 2 раза без потери качества рисованные изображения, например, изображения в стиле аниме или арт, а также позволяет устранять шум на изображении (преимущественно артефакты, возникающие при сжатии JPEG). Также есть поддержка фотографий (рисунок 1.18).

Существенным недостатком данного сервиса является его узкая специализация. Хоть разработчики Waifu2x и утверждают, что имеется поддержка работы с обычными фотографиями, но программа обрабатывает фотографии не самым лучшим образом – изображения выглядят нереалистично, словно нарисованные.

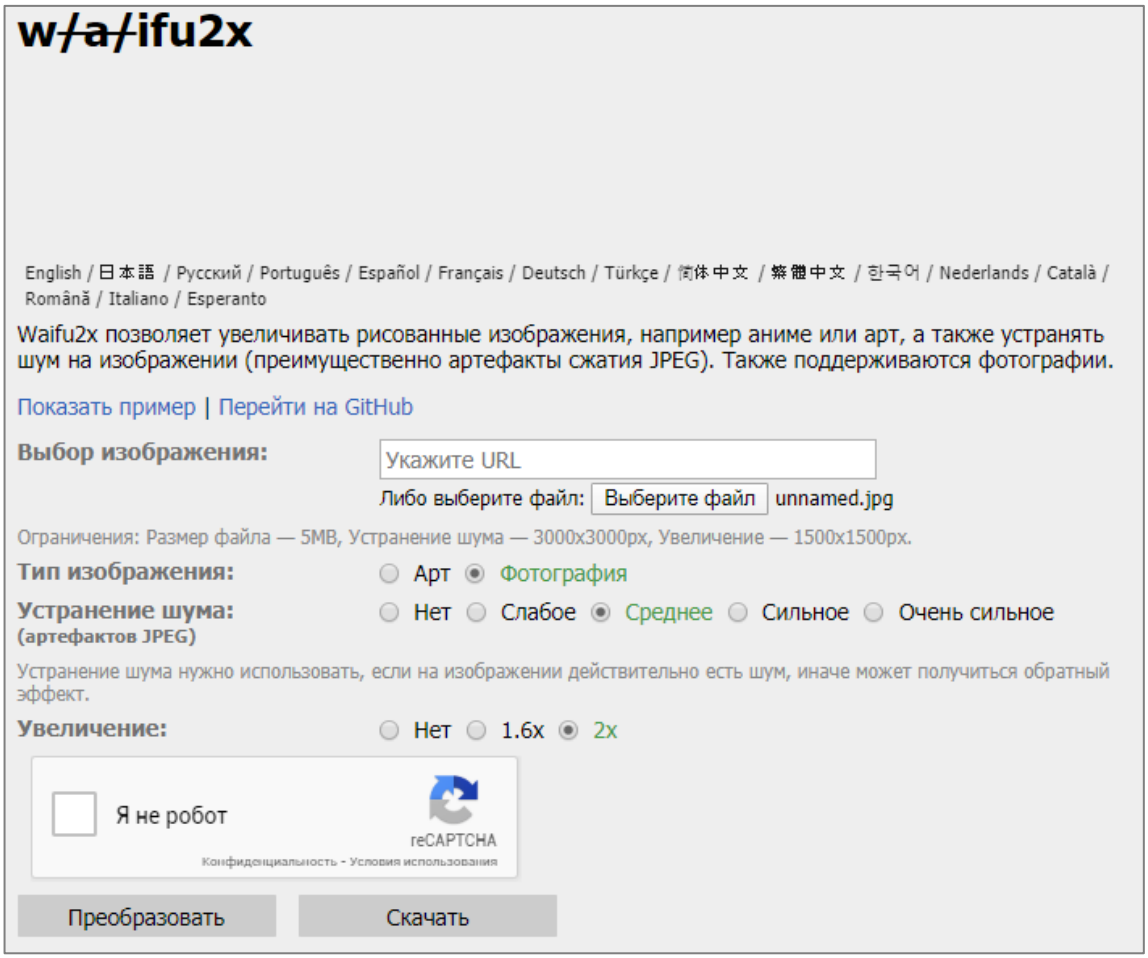


Рисунок 1.18 – Онлайн-сервис Waifu2x

А вот с изображениями, выполненными в стиле «Арт», или нарисованными изображениями сервис справляется достойно, в большинстве случаев, после обработки изображения, никаких шумов или артефактов не возникает.

Сервис является бесплатным, а также имеет открытый исходный код, что позволяет использовать его сторонним разработчикам в своих целях, например, с целью дальнейшего улучшения.

1.3 Выводы по первому разделу

В данном разделе были рассмотрены существующие методы повышения разрешения изображений без потери качества, алгоритмы их работы, а также проекты (приложения), которые используют для решения задачи сверхразрешения один из перечисленных методов.

Использование интерполяционных методов в большинстве случаев влечет за собой появление на увеличенном изображении различных артефактов, что говорит об узкой специализации таких методов. Нейросетевые же методы, основанные на сверточных сетях, могут решить проблему возникновения артефактов, но для эффективности нейронных сетей, на которых строятся эти методы, требуется большой набор данных.

Таким образом, исходя из недостатков рассмотренных методов, в данной работе будет использоваться один из нейросетевых методов сверхразрешения – генеративно-состязательный подход, который позволяет обучать нейронные сети на относительно небольшом наборе изображений, а также показывающий лучшие результаты, по сравнению с альтернативными методами повышения разрешения.

В данной работе будет разработана нейронная сеть для решения задачи повышения разрешения без потери качества. Подготовка данных, архитектура сети, а также алгоритм ее обучения и тестирования будут реализованы с помощью языка программирования Python 3.7 и библиотеки глубокого обучения PyTorch 1.2.0.

2 СВЕРХРАЗРЕШЕНИЕ ИЗОБРАЖЕНИЙ

2.1 Постановка задачи сверхразрешения изображений

Пусть X – множество изображений низкого разрешения, Y – множество изображений высокого разрешения, f^* – целевая функция, которая отображает множество X на множество Y, $f^*: X \to Y$. Значения целевой функции f^* известны только на конечном множестве пар прецедентов (x_i, y_i) – обучающей выборке. Запись $f^*(x_k) = y_k$ означает, что изображению из множества X соответствует HR изображение из множества Y.

Необходимо восстановить функциональную зависимость между изображениями низкого разрешения и изображениями высокого разрешения, построить такой алгоритм $A: X \to Y$, обладающий следующими свойствами:

– на обучающей выборке A должен производить правильные ответы;

– A должен обладать обобщающей способностью, то есть приближать целевую функцию не только на объектах обучающей выборки, но и на всем множестве X.

2.2 Исходные данные и их подготовка

Набор данных для обучения состоит из пар «изображение низкого разрешения – изображение высокого разрешения».

В качестве исходных данных для обучения нейронной сети использовалось несколько публичных наборов данных из сети Интернет:

– DIV2K – представляет собой 1600 пар изображений для обучения, 200 пар изображений для валидации и 200 пар изображений для тестирования;

– Flickr1024 – набор данных, состоящий из 1024 изображений высокого разрешения;

– InStereo2K – содержит 2050 изображений высокого разрешения;

– ImageNet – набор данных, содержащий 100000 классов, на каждый из которых приходится порядка 1000 изображений (для поставленной задачи было взято 5000 изображений различных классов).

Изображения из набора данных DIV2K уже подготовлены изначально, и дополнительная обработка для них не требуются.

Для изображений из наборов Flickr1024, InStereo2K и ImageNet соответствующие изображения низкого разрешения отсутствовали, поэтому для использования этих данных в качестве обучающей выборки необходимо было получить уменьшить исходные изображения в 4 раза для получения пар «изображения высокого разрешения – изображение низкого разрешения».

Во время подготовки данных была написана программа, которая уменьшала разрешение изображение в 4 раза с помощью случайно выбранного метода интерполяции. Таким образом, был получен набор данных, состоящий из 10000 пар изображений низкого и высокого разрешения.

2.3 Архитектура нейронной сети

Генеративно-состязательная нейронная сеть состоит из двух нейронных сетей: генератор (основная), который будет обучаться качественно повышать разрешение входных изображений низкого качества, и дискриминатор, который будет «говорить» генератору насколько хорошо или плохо ему это удается на данном этапе обучения.

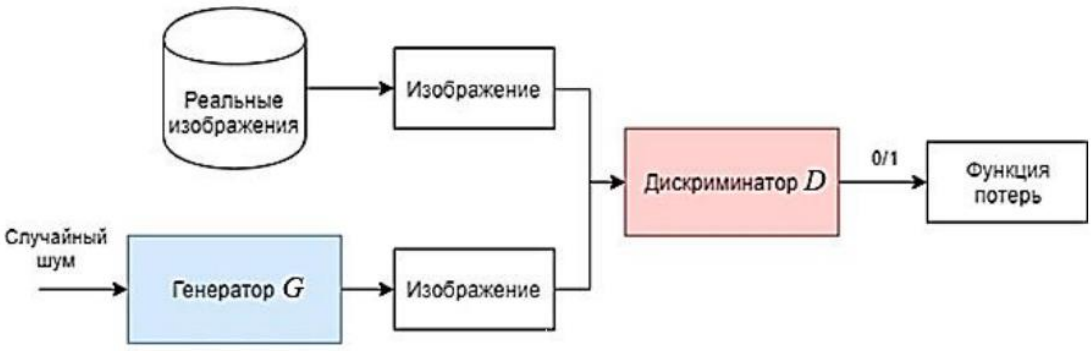


Рисунок 2.1 – Классическая генеративно-состязательная сеть

Согласно схеме, приведенной на рисунке 2.1, дискриминатор D должен быть предварительно обучен с помощью реальных изображений, которым

присвоен класс 1 («реальное изображение»). Второй этап работы алгоритма заключается в создании искусственных изображений генератором G на основе случайного шума, форма которого должна меняться в процессе обучения.

Из-за локального характера структуры изображения (близкие пиксели коррелируют, далекие – нет) многослойная сверточная сеть является эталонным выбором в качестве архитектуры для классификации изображений. Поэтому для работы с изображениями в роли дискриминатора выступает многослойная сверточная сеть, у которой последние слои являются полносвязными. Это нужно для того, чтобы дискриминатор выдавал на выходе вероятность принадлежность к одному из двух классов (число).

Модель генеративно-состязательной нейронной сети состоит из сверточных слоев, принцип работы которых рассмотрен в п. 1.1.2, слоев повышения дискретизации, которые подразумевают под собой применение интерполяционных методов к изображению, а также из слоев пакетной нормализации и остаточных блоков, которые будут рассмотрены далее.

2.3.1 Пакетная нормализация

Существует два подхода к реализации алгоритма градиентного спуска для обучения нейронных сетей: стохастический и пакетный.

Стохастический градиентный спуск – реализация, в которой на каждом шаге работы алгоритма из множества объектов обучающей выборки случайным образом выбирается один образец (изображение).

Пакетный градиентный спуск – реализация алгоритма, в котором на каждой итерации обучающая выборка просматривается целиком, и только после этого изменяются веса модели.

Существует «середина» между этими двумя подходами – когда просматривается только некоторое подмножество обучающей выборки фиксированного размера. Такие подмножества принято называть пакетами, партиями или батчами.

В 2015 году Кристиан Жегеды и Сергей Йоффе разработали для использования в нейронных сетях специальный тип слоев – слои пакетной нормализации (batch normalization). Авторы предложили выполнять нормализацию данных не только перед подачей во входной слой нейронной сети, но и перед каждым ее слоем. Нормализация выполняется отдельно для каждого пакета данных. Таким образом, пакетная нормализация позволяет повысить качество обучения глубоких нейронных сетей.

Пусть имеется пакет B, содержащий n образцов входных данных из обучающей выборки $x_i: B = \{x_1, \dots, x_m\}$. Тогда нормализованные значения $\widehat{x_i}$ рассчитываются по следующей формуле:

$$\widehat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}, \tag{2.1}$$

где μ_B – среднее значение данных в пакете;

σ_B^2 – дисперсия;

ε – выравнивающая константа, исключающая деление на ноль.

Далее, чтобы сохранить выразительность метода, выполняется сдвиг и масштабирование:

$$y_i = \gamma \widehat{x_i} + \beta, \tag{2.2}$$

где y_i – результирующее значение;

γ и β – параметры нормализации, которые определяются в процессе обучения.

Пакетная нормализация реализуется в виде слоев пакетной нормализации, которые могут быть вставлены в необходимое место нейронной сети, в том числе несколько раз. Дополнительными преимуществами использования слоев пакетной нормализации являются сокращение времени обучения и стабильность обучения, в т. ч. снижение переобучения [21].

2.3.2 Остаточный блок

В нейронных сетях добавление дополнительных слоев не всегда улучшает эффективность работы сети. В худшем случае дополнительные слои

будут просто передавать данные последующим слоям в неизменном виде. Иногда слишком глубокие сети не способны даже переобучиться (запомнить обучающий набор, но относительно плохо работать на примерах из тестовой выборки) [20].

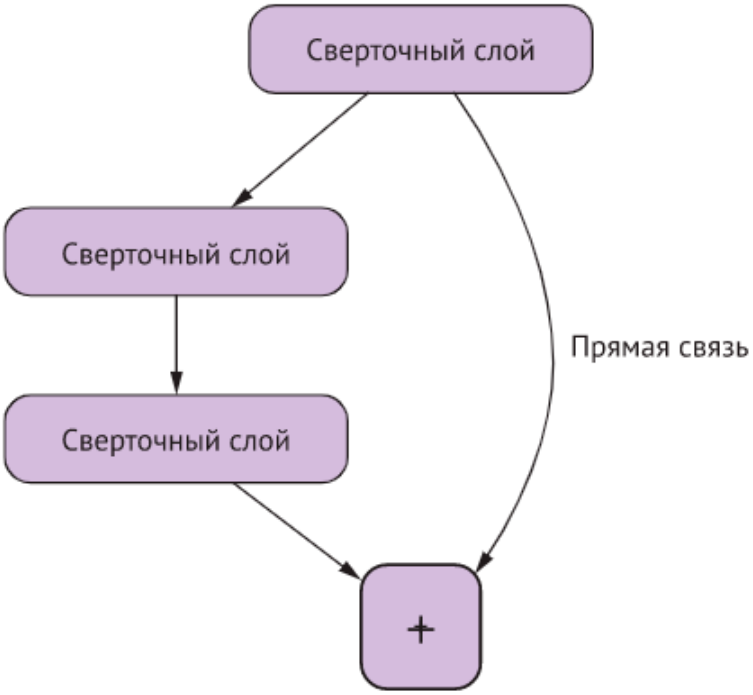


Рисунок 2.2 – Структура остаточного блока

Избежать подобной проблемы позволяют остаточные сети. Идея остаточной сети заключается в упрощении того, что пытается изучить дополнительные слой. Если первые несколько слоев хорошо справляются с изучением некоторой закономерности, то можно заставить дополнительный слой сосредоточиться на заполнении пробела между тем, что изучили предыдущие слои, и целевым значением. Этот пробел представляет собой остаток, откуда и пошло название сети.

Для реализации такой сети, мы суммируем входные данные дополнительных слоев с их выходными данными. Соединение предыдущего слоя сети со слоем, выполняющим суммирование, называется прямой связью (skip connection). Обычно остаточные сети организуются в небольшие блоки – остаточные блоки (рисунок 2.2), каждый из которых содержит несколько слоев и прямую связь.

2.4 Функции потерь

В большинстве обучающих сетей ошибка рассчитывается как разница между фактическим выходным значением y и прогнозируемым значением $\hat{y}$. Функция, используемая для вычисления этой ошибки, известна как функция потерь, также часто называемая функцией ошибки или затрат.

В генеративно-состязательных нейронных сетях для обучения используются несколько функций потерь. Это обусловлено тем, что цель обучения у генератора и дискриминатора разная: генератору необходимо порождать наиболее реалистичные данные, а дискриминатору – уметь такие данные классифицировать, т. е. отличать от подделки.

2.4.1 Функция потерь восприятия

В задаче сверхразрешения для обучения сети используют функцию потерь восприятия (perceptual loss function), которая является взвешенной суммой двух функций:

$$l = l_{CL} + 0.001 \cdot l_{AL}, \tag{2.3}$$

где l_{CL} – функция потерь контента (content loss function);

l_{AL} – состязательная функция потерь (adversarial loss function).

2.4.2 Функция потерь контента

Функция потерь контента – это среднеквадратичная ошибка между значениями каждого пикселя реального изображения и значениями каждого пикселя сгенерированного изображения. Значение этой функции вычисляется как разница между двумя изображениями по следующей формуле:

$$l_{CL} = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(I_{i,j}^{HR} - G(I^{LR})_{i,j}\right)^2, \tag{2.4}$$

где m – ширина изображения;

n – высота изображения;

i, j – координаты пикселя;

I^{HR} – реальное изображение высокого разрешения;

I^{LR} – изображение низкого разрешения;

$G(I^{LR})$ – изображения высокого разрешения, сгенерированное нейросетью.

2.4.3 Состязательная функция потерь

Состязательная функция потерь вычисляется, используя вероятности, полученные от дискриминатора. Значение этой функции потерь вычисляется следующим образом:

$$l_{AL} = \sum_{i=1}^{N} -\ln D(G(I_i^{LR})), \tag{2.5}$$

где N – количество изображений в пакете;

I^{LR} – изображение низкого разрешения;

$G(I^{LR})$ – изображения высокого разрешения, сгенерированное нейросетью;

$D(G(I^{LR}))$ – выход дискриминатора, вероятность (реальное или сгенерированное).

2.4.4 Среднеквадратичная ошибка

Для вычисления ошибки дискриминатора используется функция потерь, называемая среднеквадратичной ошибкой (mean squared error – MSE). Значение этой функции потерь вычисляется следующим образом:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2, \tag{2.6}$$

где N – количество обработанных нейронной сеть изображений;

p_i – реальный выход нейронной сети;

y_i – верный (желаемый) выход нейронной сети.

2.5 Обучение сети

Обучение нейронной сети производится с помощью метода обратного распространения ошибки [28]. Основная цель – обучение нейронной сети до

момента достижения баланса между способностью сети выдавать верный ответ на входные данные из обучающей выборки (запоминания), и способностью выдавать правильные результаты в ответ на входные данные, схожие, но неидентичные тем, что были использованы при обучении (принцип обобщения).

Обучение сети методом обратного распространения ошибки состоит из двух этапов:

1) прямой проход – подача данных на вход сети с их последующим распространением в направлении выходов;

2) обратный проход – вычисление и обратное распространение ошибки, а также корректировка весов модели.

Рассмотрим алгоритм обратного распространения ошибки для многослойного персептрона, имеющего входной слой $k = 0$, несколько скрытых слоев $k = 1,2, \dots, K-1$ и выходной слой $k = K$.

Нейроны входного слоя не выполняют математических преобразований, а лишь передают входные сигналы нейронам первого слоя. Будем полагать, что каждый k-ый слой содержит H_k нейронов. Таким образом, персептрон имеет $N = H_0$ входов и $M = H_K$ выходов. В алгоритме будем использовать следующие обозначения: i – порядковый номер нейрона k-го слоя; j – порядковый номер нейрона $(k-1)$-го слоя; l – порядковый номер нейрона $(k+1)$-го слоя; η – скорость обучения; f – функция активации.

Шаг 1. Инициализация синаптических весов и смещений. В циклах по $k = 1,2, \dots, K; i = 1, 2, \dots, H_k; j = 0, 1, 2, \dots, H_{k-1}$ генератор случайных чисел присваивает синаптическим весам и смещениям $w_{ij}^{(k)}$ малые величины, например, из диапазона от -1 до 1.

Шаг 2. Открытие цикла по $q = 1,2, \dots, Q$. Представление из обучающего множества примеров очередного входного вектора $X_q = (x_1, x_2, \dots, x_N)_q$ и соответствующего ему желаемого выходного вектора $D_q = (d_1, d_2, \dots, d_M)_q$, где q – номер примера в обучающем множестве.

Шаг 3. Прямой проход. В циклах по $k = 1, 2, \dots, K; i = 1, 2, \dots, H_k$ вычисляются выходные сигналы i-го нейрона в k-ом слое:

$$y_i^{(k)} = f\left(\sum_{j=0}^{H_{k-1}} w_{ij}^{(k)} y_j^{(k-1)}\right), \tag{2.7}$$

где $y_j^{(0)} = x_j; x_0 = 1; y_0^{(k-1)} = 1; y_i = y_i^{(K)}$ – выходные сигналы персептрона.

Шаг 4. Обратный проход. В циклах по $k = K, K-1, \dots, 1; i = 1, 2, \dots, H_k;$ $j = 0, 1, 2, \dots, H_{k-1}$ вычисляются синаптические веса на новой эпохе:

$$w_{ij}^{(k)}(t+1) = w_{ij}^{(k)}(t) + \Delta w_{ij}^{(k)}, \tag{2.8}$$

$$\Delta w_{ij}^{(k)} = \eta \delta_i^{(k)} y_j^{(k+1)}. \tag{2.9}$$

Для нейронов выходного слоя $k = K$:

$$\delta_i^{(K)} = y_i(1 - y_i)(d_i - y_i). \tag{2.10}$$

Для остальных скрытых слоев:

$$\delta_i^{(k)} = y_i^{(k)}\left(1 - y_i^{(k)}\right) \sum_{l=1}^{H_{k+1}} \delta_l^{(k+1)} w_{li}^{(k+1)}. \tag{2.11}$$

Шаг 5. Закрытие цикла по q.

Шаг 6. Повторение шагов 2–5 необходимое количество раз. Векторы обучающих примеров X_q и D_q на шаге 2 алгоритма обычно представляются последовательно от первого до последнего, т. е. $q = 1, 2, \dots, Q$, где Q – общее количество примеров.

После того как для каждого обучающего примера будут скорректированы весовые коэффициенты персептрона, т. е. шаги 2–4 будут повторены Q раз, на шаге 6 алгоритма вычисляется значение функции потерь ε , усредненное по всем обучающим примерам.

В данной работе ε рассчитывается по формулам 2.3 и 2.6 для генератора и дискриминатора соответственно.

Итерационный процесс, задаваемый шагом 6, заканчивается после того, как ошибка ε достигнет заданной величины либо когда будет достигнуто предельное количество эпох обучения.

После обучения предполагается лишь подача на вход сети данных и распространение их в направлении выходов. При этом, если обучение сети может являться довольно длительным процессом, то непосредственное вычисление результатов обученной сетью происходит очень быстро [4].

2.6 Оптимизатор Adam

Поиск оптимального значения используемой функции потерь осуществляется с помощью оптимизационного метода – Адам (Adam). Алгоритм оптимизации Adam является модификацией стохастического градиентного спуска, который в последнее время получил широкое распространение в использовании в глубоких нейронных сетях.

Основные преимущества оптимизатора:

– простая реализация;

– вычислительная эффективность;

– небольшие требования к памяти;

– гиперпараметры требуют небольшой настройки и имеют интуитивно понятную интерпретацию.

Адам отличается от классического стохастического градиентного спуска: стохастический градиентный спуск использует единую скорость обучения для всех весов модели, а также скорость обучения не изменяется во время тренировки, в отличие от Адама. При использовании Адама скорость обучения определяется индивидуально для каждого веса сети и адаптируется отдельно в процессе обучения сети. Метод вычисляет индивидуальные адаптивные скорости обучения для различных параметров, используя оценки первого и второго моментов градиентов [4].

Оптимизатор Адам является объединением преимуществ двух других модификаций стохастического градиентного спуска (рисунок 2.3).

1. Адаптивный градиентный алгоритм (AdaGrad), который поддерживает скорость обучения по параметру, которая улучшает производительность при проблемах с разреженными градиентами (например, проблемы с естественным языком и компьютерным зрением).

2. Среднеквадратичное распространение (RMSProp), который также поддерживает скорости обучения по каждому параметру, которые адаптированы на основе среднего значения последних величин градиентов для веса (например, как быстро оно изменяется). Это означает, что алгоритм хорошо справляется с нестационарными задачами (например, с шумом).

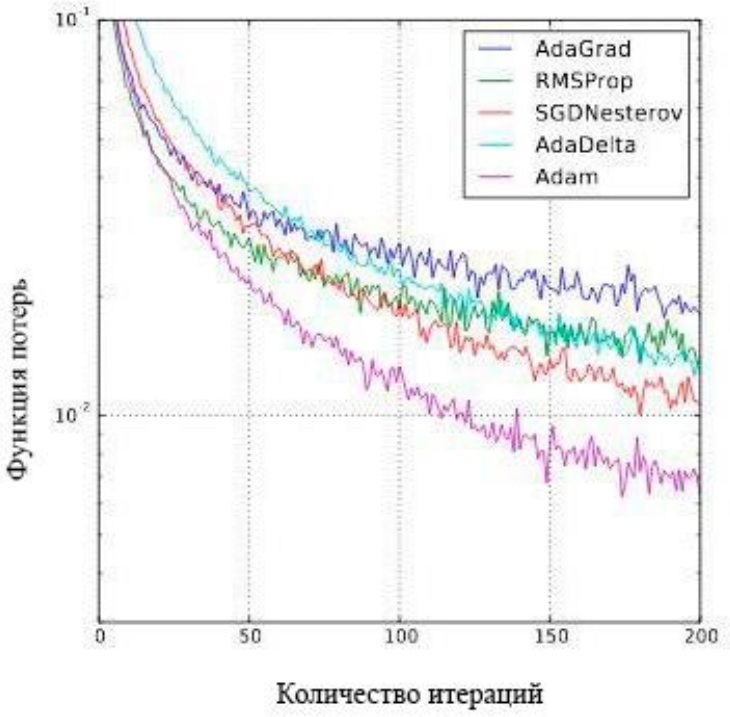


Рисунок 2.3 – Сравнение скорости сходимости
Адама и других оптимизаторов

Еще одним характерным преимуществом данного оптимизационного алгоритма является его способность выходить из так называемых седловых точек – точек объединения нескольких локальных минимумов невыпуклой функции (рисунок 2.4).

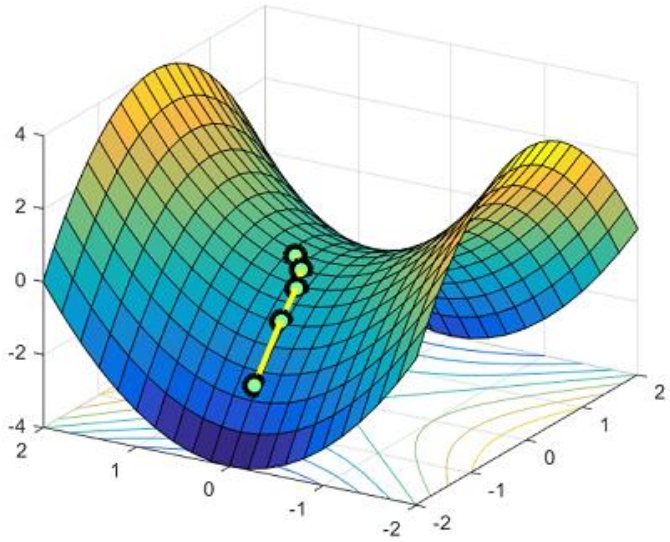


Рисунок 2.4 – «Побег» из седловой точки функции

Алгоритм работы:

1) инициализация начальных моментов: $M_0 = 0, R_0 = 0$;

2) для каждого веса W_t, $t = 1, \dots, T$ выполнить:

– оценку первого момента:

$$M_t = \beta_1 M_{t-1} + (1 - \beta_1)\nabla L_t(W_{t-1}); \tag{2.12}$$

– оценку второго момента:

$$R_t = \beta_2 R_{t-1} + (1 - \beta_2)\nabla L_t(W_{t-1})^2; \tag{2.13}$$

– коррекцию смещения первого момента:

$$\widehat{M_t} = \frac{M_t}{(1 - \beta_1)^{\,t}}; \tag{2.14}$$

– коррекцию смещения второго момента:

$$\widehat{R_t} = \frac{R_t}{(1 - \beta_2)^t}; \tag{2.15}$$

– обновление:

$$W_t = W_{t-1} - \alpha \frac{\widehat{M_t}}{\sqrt{\widehat{R_t} + \varepsilon}}. \tag{2.16}$$

3) вернуть веса W_T.

Параметры конфигурации Адама:

1) α – величина, которая называется скоростью обучения или размером шага. Пропорция веса обновляется (например, 0.001). Большие значения

(например, 0,3) приводят к более быстрому начальному обучению до обновления скорости. Меньшие значения (например, $1.0E-5$) замедляют обучение прямо во время тренировки;

2) β_1 – показатель экспоненциальной скорости затухания для первого момента (например, 0.9);

3) β_2 – показатель экспоненциального спада для оценок второго момента (например, 0.999). Это значение должно быть близко к 1.0 в случае проблем с разреженным градиентом (например, проблемы НЛП и компьютерного зрения);

4) ε – это очень небольшое число, позволяющее предотвратить любое деление на ноль при реализации (например, $10E-8$).

2.7 Метрики качества

2.7.1 Метрика $PSNR$

Пиковое отношение сигнала к шуму $PSNR$ (англ. Peak Signal-to-Noise Ratio) характеризует степень зашумленности двух изображений относительно друг друга, учитывая максимальное возможное значение входного сигнала и мощность шума (рисунок 2.5).

Рисунок 2.5 – Значения $PSNR$ для зашумленного
и сжатого изображения соответсвенно

$PSNR$ принято измерять в логарифмической шкале в децибелах, поскольку многие сигналы имеют широкий диапазон возможных значений. $PSNR$ рассчитывается с помощью среднеквадратичного отклонения σ, которое для двух черно-белых изображений I и K размера $m \times n$, одно из которых является эталонным изображением, не содержащим шума, а другое – его приближением, вычисляется так:

$$\sigma = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |I(i,j) - K(i,j)|^2, \tag{2.17}$$

где $I(i,j)$ – значение пикселя с координатами (i,j) эталонного изображения;

$K(i,j)$ – значение пикселя с координатами (i,j) сравниваемого изображения;

m и n – ширина и высота изображений соответственно.

$PSNR$ определяется следующим образом:

$$PSNR = 20 \log_{10} \left(\frac{255}{\sqrt{\sigma}} \right). \tag{2.18}$$

Для цветных изображений пространства RGB (3 компоненты на один пиксель) применяется такое же определение $PSNR$, но среднеквадратичное отклонение σ вычисляется по всем трем компонентам: значениям красного канала R, зеленого канала G, синего канала B:

$$\sigma = \frac{1}{3mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \sum_{l=0}^{2} |x_l(i,j) - y_l(i,j)|^2, \tag{2.19}$$

где $I_l(i,j)$ – значение пикселя l-го канала с координатами (i,j) эталонного изображения;

$K_l(i,j)$ – значение пикселя l-го канала с координатами (i,j) сравниваемого изображения;

m и n – ширина и высота изображений соответственно.

Типичные значения $PSNR$ при обработке изображений с целью изменения его разрешения находятся в диапазоне от 30 до 40 dB. Чем больше

значение $PSNR$, тем менее зашумленным является обработанное изображение (относительно эталонного).

Данная метрика аналогична среднеквадратичному отклонению, однако за счет логарифмического масштаба шкалы пользоваться ей гораздо удобнее. Следует отметить, что высокое значение метрики $PSNR$ не всегда гарантирует хорошее качество восстановленного изображения, поскольку визуальное восприятие образов человеческим глазом обладает нелинейным поведением. При наличии некоторых шумов в изображении оценка может оставаться такой же, хотя качество изображения при этом значительно меняется.

2.7.2 Метрика $SSIM$

В отличие от предыдущей метрики $SSIM$ (индекс структурной схожести изображений) учитывает искажение контрастности и яркости, а также степень коррелированности соответствующих значений пикселей между двумя изображениями.

Метрика $SSIM$ является более универсальной, поскольку она демонстрирует не только степень структурной схожести обработанного изображения относительно эталонного, но и учитывает различные виды геометрических искажений. Данная метрика не привязана к специфике изображения и искажениям, содержащимся в нем, а основывается на статическом анализе отдельных частей входного сигнала и дальнейшем сравнении полученных результатов со значениями эталонного изображения (рисунок 2.6).

Рисунок 2.6 – Значения $SSIM$ для оригинального, зашумленного и контрастного изображений соответственно

Оценка сводится к определению степени сходства соответствующих частей сравниваемых изображений по трем составляющим:

1) $L(x,y) = \frac{2\mu_x\mu_y+K}{\mu_x^2+\mu_y^2+K}$ – яркость (значения математического ожидания пикселей изображений);

2) $C(x,y) = \frac{2\sigma_x\sigma_y+K}{\sigma_x^2+\sigma_y^2+K}$ – контраст (значения среднеквадратичного отклонения пикселей изображений);

3) $S(x,y) = \frac{\sigma_{xy}+K}{\sigma_x\sigma_y+K}$ – структура (степень коррелированности пикселей изображений).

В вышеприведенных формулах:

x – матрица значений пикселей оригинального изображения;

y – матрица значений пикселей оцениваемого изображения;

μ_x – среднее арифметическое значение для оригинального изображения (μ_y – для оцениваемого) размером M пикселей:

$$\mu_x = \frac{1}{M}\sum_{i=1}^{M} x_i\,; \tag{2.20}$$

где σ_x^2 – дисперсия для оригинального изображения (σ_y^2 – для оцениваемого) размером M пикселей:

$$\sigma_x^2 = \frac{1}{M}\sum_{i=1}^{M}(x_i-\mu_x)^2; \tag{2.21}$$

где σ_{xy} – корреляционный момент участков сравниваемых изображений:

$$\sigma_{xy} = \frac{1}{M}\sum_{i=1}^{M}(x_i-\mu_x)\big(y_i-\mu_y\big); \tag{2.22}$$

где K = 0.01 – выравнивающий коэффициент, предотвращающий деление на число, близкое к нулю.

Используя данные обозначения, формула для вычисления $SSIM$ будет выглядеть следующим образом:

$$SSIM = \frac{(2\mu_x\mu_y + K)(2\sigma_{xy} + K)}{(\mu_x^2 + \mu_y^2 + K)(\sigma_x^2 + \sigma_y^2 + K)}. \quad (2.23)$$

Значения метрики $SSIM$ лежат в диапазоне от –1 до 1. Для идентичных изображений величина метрики $SSIM$ равна 1. Метрика $SSIM$ дает достаточно адекватную для человеческого восприятия оценку качества изображений.

2.8 Функции активации

Функция активации (активационная функция) – функция, преобразующая выходной сигнал искусственного нейрона. В качестве аргумента функции используется сигнал y, получаемый на выходе входного сумматора $\sum$. Функции активации являются одним из способов нормализации данных. То есть, если на входе мы получаем большое число, то пропуская его через функцию активации, можно получить число в нужном диапазоне.

В скрытых слоях нейронной сети в качестве функций активации используются полулинейная функция ReLU (рисунок 2.7) и полулинейная функция с «утечкой» LeakyReLU (рисунок 2.8).

2.8.1 Функция активации ReLU

Функция активации ReLU имеет следующий вид:

$$f(y) = \max(0, y). \quad (2.24)$$

Данная функция активации имеет несколько преимуществ. Во-первых, ReLU достаточно проста в реализации. Например, она может быть реализована с помощью порогового преобразования матрицы активаций в нуле, в то время как сигмоидальная функция и функция гиперболического тангенса при вычислении выходных значений требуют выполнение ресурсоемких операций, таких как возведение в степень. Во-вторых, использование функции активации ReLU значительно увеличивает скорость сходимости стохастического градиентного спуска (в некоторых случаях до 5 раз) по сравнению с сигмоидальной функцией активации и гиперболическим тангенсом. Считается, что

это обусловлено отсутствием насыщения данной функции и линейным характером.

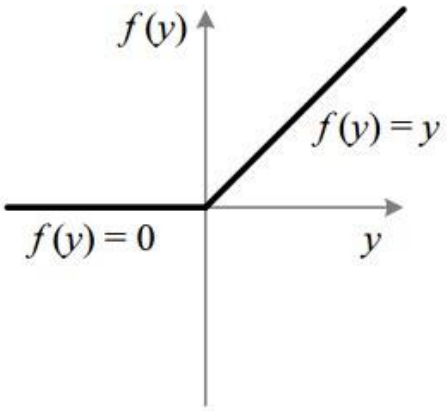


Рисунок 2.7 – Активационная функция ReLU

Недостатком активационной функции ReLU является недостаточная надежность (в процессе обучения они могут выходить из строя). Например, большой градиент, проходящий через ReLU, может привести к такому обновлению весов, что данный нейрон никогда больше не активируется. Если это произойдет, то, начиная с данного момента, градиент, проходящий через этот нейрон, всегда будет равен нулю. Соответственно, данный нейрон будет необратимо выведен из строя. При большой скорости обучения может оказаться, что 30-40% активационных слоев ReLU «умерли». Эта проблема решается посредством подбора скорости обучения нейронной сети.

2.8.2 Функция активации LeakyReLU

Функция активации LeakyReLU представляет собой модификацию функции ReLU, которая лишена вышеописанной проблемы с затуханием градиентов. Обычный ReLU на интервале $y < 0$ дает на выходе 0, в то время как LeakyReLU имеет на этом интервале небольшое отрицательное значение (угловой коэффициент около 0.01).

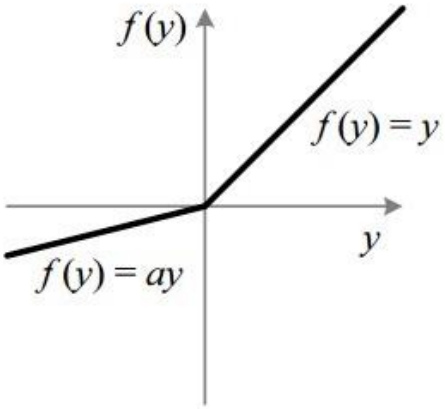


Рисунок 2.8 – Активационная функция LeakyReLU

Функция LeakyReLU имеет следующий вид:

$$f(y) = \begin{cases} 0.01y, y < 0, \\ y, y > 0. \end{cases} \quad (2.25)$$

2.8.3 Функция активации Tanh

Функция активации Tanh – это гиперболический тангенс:

$$f(y) = \mathrm{Tanh}(y) = \frac{\sinh(y)}{\cosh(y)} = \frac{2}{1 + e^{-2y}} - 1. \quad (2.26)$$

Гиперболический тангенс очень похож на сигмоидальную функцию и обладает теми же преимуществами.

Во-первых, данная функция активации является дифференцируемой, что позволяет использовать для обучения сети метод обратного распространения ошибки.

Во-вторых, гиперболический тангенс является нелинейной функцией, что хорошо подходит для комбинации слоев нейронной сети.

В-третьих, диапазон значений активационной функции Tanh – $(-1; 1)$, поэтому нет смысла беспокоиться о том, что функция перегрузится от больших значений (рисунок 2.9).

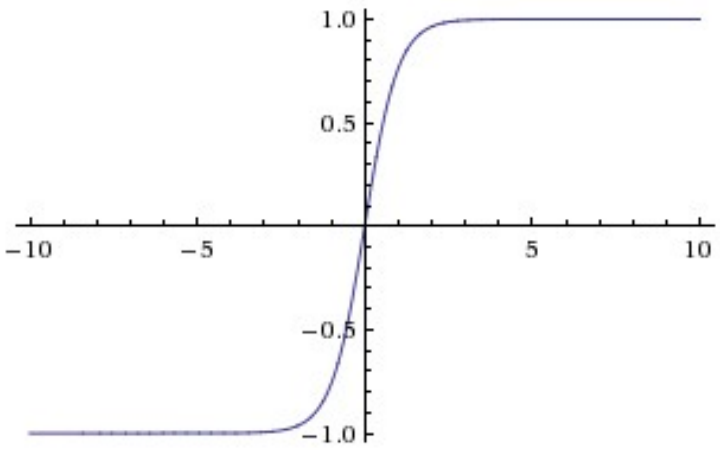


Рисунок 2.9 – Активационная функция Tanh

2.8.4 Функция активации Sigmoid

Сигмоида (сигмоидальная или логистическая функция активации) используется в том случае, когда нужно привести выход нейрона к диапазону от 0 до 1, т. е. получить некоторое значение вероятности (рисунок 2.10). Данная функция активации используется на выходе последнего слоя в дискриминаторе для того, чтобы определить вероятность принадлежности поданного на вход изображения к одному из двух классов.

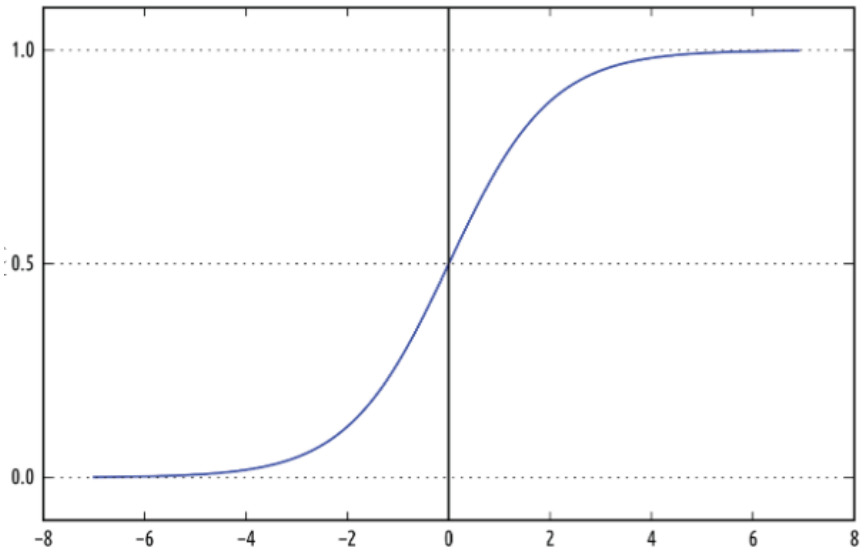


Рисунок 2.10 – Сигмоидальная функция активации

Сигмоидальная функция активации имеет следующий вид:

$$f(y) = \frac{1}{1 + e^{-y}}. \tag{2.27}$$

2.9 Выводы по разделу

В данном разделе была сформулирована и поставлена задача сверхразрешения изображений, имеющиеся для нее исходные данные и их подготовка, а также архитектура генеративно-состязательной сети. Были рассмотрены алгоритм обучения таких сетей (метод обратного распространения ошибки), функции потерь, используемые в поставленной задаче и алгоритмы оптимизации таких функций.

Для построения модели эффективной нейронной сети были рассмотрены и изучены наиболее популярные и подходящие функции активации. Также в данном разделе приведены метрики качества, позволяющие оценить эффективность работы разработанной нейронной сети.

3 РЕАЛИЗАЦИЯ СЕТИ И ПРОВЕРКА НА ТЕСТОВЫХ ДАННЫХ

3.1 Архитектура сети

Архитектура сети состоит из двух частей (рисунок 3.1).

1. Генератор – сеть, которая преобразует с помощью серии конволюций изображение низкого разрешения размерностью $64 \times 64 \times 3$ в изображение высокого разрешения размерностью $256 \times 256 \times 3$.

2. Дискриминатор – сеть, которая получает на вход изображение высокого разрешения и пытается идентифицировать его как реальное изображение (изображение из реального набора данных) или как сгенерированное изображение.

3.1.1 Архитектура генератора

Генераторная сеть представляет собой нейронную сеть с глубокими свертками. Генератор содержит следующие блоки:

– предварительный блок;

– остаточный блок;

– пост-остаточный блок;

– блок повышения дискретизации;

– последний сверточный блок.

Рассмотрим, что из себя представляет каждый такой блок.

1. Предварительный блок состоит из одиночного слоя двумерной свертки и следующей за ним функции активации ReLU (таблица 3.1).

Таблица 3.1 – Предварительный блок

Название слоя	Гиперпараметры	Вход	Выход
Слой двумерной свертки	Признаки = 64, Размер ядра = 3, Шаг = 1, Отступ = 1, Активация = ReLU	(64, 64, 3)	(64, 64, 64)

2. Остаточный блок состоит из двух сверточных слоев, после каждого из которых расположен слой пакетной нормализации (таблица 3.2).

Таблица 3.2 – Остаточный блок

Название слоя	Гиперпараметры	Вход	Выход
Слой двумерной свертки	Признаки = 64, Размер ядра = 3, Шаг = 1, Отступ = 1, Активация = ReLU	(64, 64, 64)	(64, 64, 64)
Слой нормализации		(64, 64, 64)	(64, 64, 64)
Слой двумерной свертки	Признаки = 64, Размер ядра = 3, Шаг = 1, Отступ = 1, Активация = ReLU	(64, 64, 64)	(64, 64, 64)
Слой нормализации		(64, 64, 64)	(64, 64, 64)
Объединяющий слой		(64, 64, 64)	(64, 64, 64)

В конце располагается объединяющий слой, который вычисляет сумму входных в блок признаков и признаков, полученных на выходе из блока.

3. Пост-остаточный блок содержит слой двумерной свертки с функцией активации ReLU и слой пакетной нормализации (таблица 3.3).

Таблица 3.3 – Пост-остаточный блок

Название слоя	Гиперпараметры	Вход	Выход
Слой двумерной свертки	Признаки = 64, Размер ядра = 3, Шаг = 1, Отступ = 1, Активация = ReLU	(64, 64, 64)	(64, 64, 64)

Продолжение таблицы 3.3

Название слоя	Гиперпараметры	Вход	Выход
Слой нормализации		(64, 64, 64)	(64, 64, 64)

4. Блок повышения дискретизации содержит один слой повышения дискретизации и один сверточный слой (таблица 3.4). Слой повышения дискретизации представляет собой простое масштабирование карт признаков с помощью билинейной интерполяции или интерполяции методом ближайшего соседа.

Таблица 3.4 – Блок повышения дискретизации

Название слоя	Гиперпараметры	Вход	Выход
Слой повышения дискретизации	Размер = 2 × 2, Метод = билинейный	(64, 64, 64)	(128, 128, 64)
Слой двумерной свертки	Признаки = 256, Размер ядра = 3, Шаг = 1, Отступ = 1, Активация = ReLU	(128, 128, 256)	(128, 128, 256)
Слой повышения дискретизации	Размер = 2 × 2, Метод = билинейный	(128, 128, 256)	(256, 256, 256)
Слой двумерной свертки	Признаки = 256, Размер ядра = 3, Шаг = 1, Отступ = 1, Активация = ReLU	(256, 256, 256)	(256, 256, 256)

5. Последний сверточный блок состоит из слоя одного сверточного слоя (таблица 3.5). функция состоит в том, чтобы привести полученные карты признаков к выходному изображению, состоящему из трех каналов.

Таблица 3.5 – Последний сверточный блок

Название слоя	Гиперпараметры	Вход	Выход
Слой двумерной свертки	Признаки = 3, Размер ядра = 9, Шаг = 1, Отступ = 1, Активация = tanh	(256, 256, 256)	(256, 256, 3)

Таким образом, используя ранее описанные блоки, можно сформировать архитектуру генерирующей сети, которая будет повышать разрешение изображений (таблица 3.6).

Таблица 3.6 – Архитектура генератора

Название блока	Гиперпараметры	Вход	Выход
Предварительный блок	Количество = 1	(64, 64, 3)	(64, 64, 64)
Остаточный блок	Количество = 16	(64, 64, 64)	(64, 64, 64)
Пост-остаточный блок	Количество = 1	(64, 64, 64)	(64, 64, 64)
Блок повышения дискретизации	Количество = 1	(64, 64, 64)	(256, 256, 256)
Последний сверточный блок	Количество = 1	(256, 256, 256)	(256, 256, 3)

3.1.2 Архитектура дискриминатора

Дискриминаторная сеть также является глубокой сверточной сетью (таблица 3.7). Она содержит 8 сверточных блоков, за которыми следуют два полносвязных слоя. Каждый сверточный блок состоит из сверточного слоя и слоя пакетной нормализации.

Полносвязные слои классифицируют извлеченные сверточными блоками карты признаков. Последний слой прогнозирует вероятность того, какому классу принадлежит изображение: реальному набору данных или поддельному набору данных.

Таблица 3.7 – Архитектура дискриминатора

Название слоя	Гиперпараметры	Вход	Выход
Слой двумерной свертки	Признаки = 64, Размер ядра = 3, Шаг = 1, Отступ = 1, Активация = LeakyReLU	(256, 256, 3)	(256, 256, 64)
Слой двумерной свертки	Признаки = 64, Размер ядра = 3, Шаг = 2, Отступ = 1, Активация = LeakyReLU	(256, 256, 64)	(128, 128, 64)
Слой нормализации		(128, 128, 64)	(128, 128, 64)
Слой двумерной свертки	Признаки = 128, Размер ядра = 3, Шаг = 1, Отступ = 1, Активация = LeakyReLU	(128, 128, 64)	(128, 128, 128)
Слой нормализации		(128, 128, 128)	(128, 128, 128)
Слой двумерной свертки	Признаки = 128, Размер ядра = 3, Шаг = 2, Отступ = 1, Активация = LeakyReLU	(128, 128, 128)	(64, 64, 128)
Слой нормализации		(64, 64, 128)	(64, 64, 128)
Слой двумерной свертки	Признаки = 256, Размер ядра = 3, Шаг = 1, Отступ = 1, Активация = LeakyReLU	(64, 64, 128)	(64, 64, 256)

Продолжение таблицы 3.7

Слой нормализации		(64, 64, 256)	(64, 64, 256)
Слой двумерной свертки	Признаки = 256, Размер ядра = 3, Шаг = 2, Отступ = 1, Активация = LeakyReLU	(64, 64, 256)	(32, 32, 256)
Слой нормализации		(32, 32, 256)	(32, 32, 256)
Слой двумерной свертки	Признаки = 512, Размер ядра = 3, Шаг = 1, Отступ = 1, Активация = LeakyReLU	(32, 32, 256)	(32, 32, 512)
Слой нормализации		(32, 32, 512)	(32, 32, 512)
Слой двумерной свертки	Признаки = 512, Размер ядра = 3, Шаг = 2, Отступ = 1, Активация = LeakyReLU	(32, 32, 512)	(16, 16, 512)
Слой нормализации		(16, 16, 512)	(16, 16, 512)
Полносвязный слой	Нейроны = 1024, Активация = LeakyReLU		
Полносвязный слой	Нейроны = 1, Активация = Sigmoid		

3.2 Обучение сети

Перед тем, как начать тренировать нейронную сеть, были подготовлены тренировочная и тестовая выборки, процесс подготовки данных подробно описан в п. 2.2.

Также были заданы гиперпараметры нейронной сети, т. е. параметры сети, которые не будут меняться в процессе ее обучения. К гиперпараметрам относятся: скорость обучения, количество эпох (эпохой называется проход всех пакетов обучающей выборки через сеть), метод оптимизации, количество остаточных блоков генератора и количество изображений в пакете.

Далее был создан загрузчик данных – объект, который делит тренировочные данные на пакеты и подготавливает изображения пакета перед непосредственной подачей на вход сети.

Обучение сети осуществлялось с помощью оптимизатора Adam, который минимизировал значение функции потерь. Используемые функции потерь для генератора и дискриминатора подробно описаны в п. 2.8.

Данные на вход сети подаются последовательно небольшими пакетами. Сначала данные подаются на вход сети генератору, который повышает разрешение входного изображения, после чего вычисляется значение функции потерь генератора для данного пакета, а затем происходит обновление весов данной подсети.

На следующем шаге на вход дискриминатору подаются реальные данные и данные, полученные на выходе генератора, после чего вычисляется значение функции потерь дискриминатора и обновление его весов. Стоит отметить, что при обучении одной подсети, веса другой подсети «замораживаются».

После завершения эпохи веса модели сохраняются. Каждую десятую эпоху сохраняются текущие значения функций потерь генератора и дискриминатора для того, чтобы можно было визуализировать процесс обучения и

сделать вывод о том, когда нейронная сеть будет завершать свое обучение. Код обучения и тестирования сети представлен в приложении 1.

По завершению обучения сети, ее работа проверяется на тестовой выборке, для которой подсчитываются значений метрик качества $PSNR$ и $SSIM$. Код для подсчета метрик на тестовой выборке представлен в приложении 2.

Схематичное описание алгоритма работы программы и алгоритма обучения сети представлены на рисунках 3.2–3.3.

Рисунок 3.2 – Алгоритм работы программы

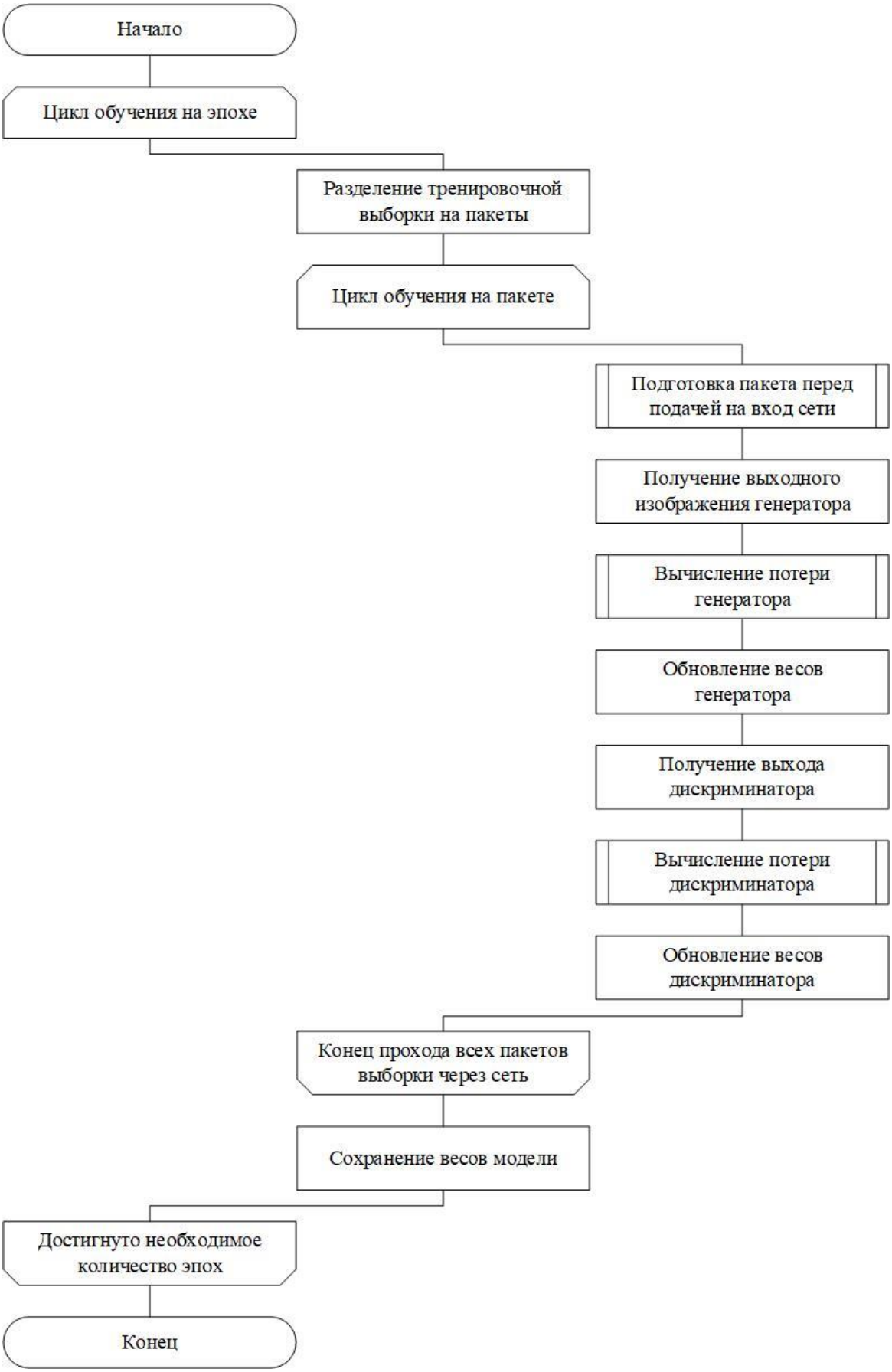


Рисунок 3.3 – Вспомогательный алгоритм «Обучение сети»

3.3 Результаты обучения

Для построения и визуализации графиков обучения нейронной сети использовалась библиотека TensorBoard. TensorBoard – это инструмент для визуализации графиков, количественных метрик выполнения графиков и дополнительных данных, которые могут помочь понять поведение нейронной сети в процессе обучения (рис. 3.4–3.5).

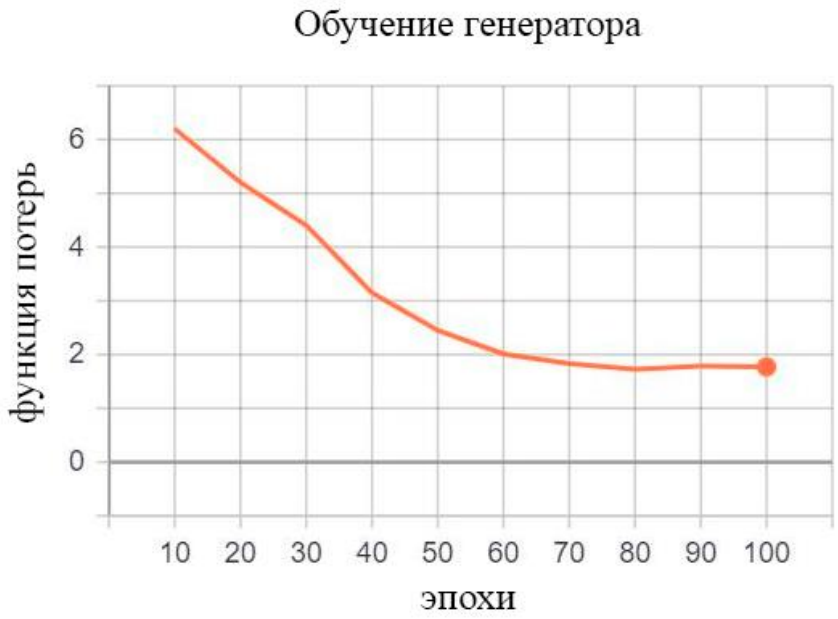


Рисунок 3.4 – График зависимости потери генератора от количества эпох

Рисунок 3.5 – График зависимости потери дискриминатора от количества эпох

3.4 Проверка на тестовых данных

Обученная модель нейронной сети была проверена на тестовой выборке. Каждое изображение тестовой выборки было подано на вход генератору, после чего были подсчитаны метрики качества относительно исходного изображения высокого разрешения. Для наглядности на рисунках 3.6–3.9 также представлены изображения в высоком и низком разрешении, а для сгенерированного изображения рассчитаны значения $PSNR$ и $SSIM$.

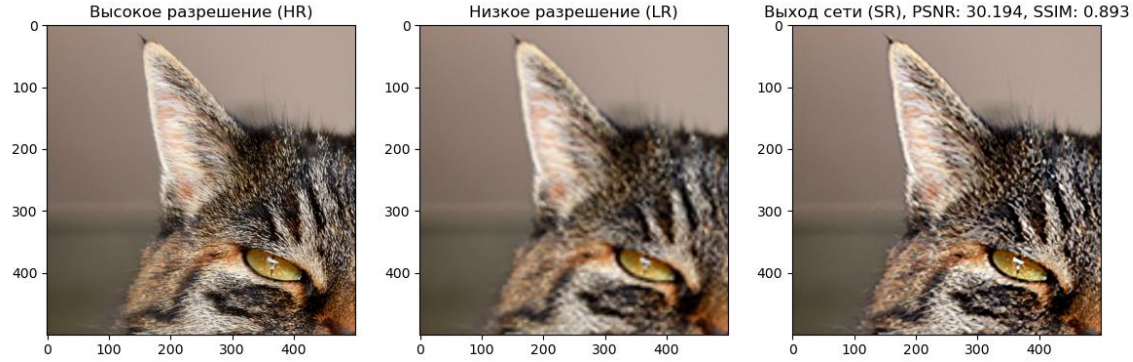


Рисунок 3.6 – Пример №1 работы сети

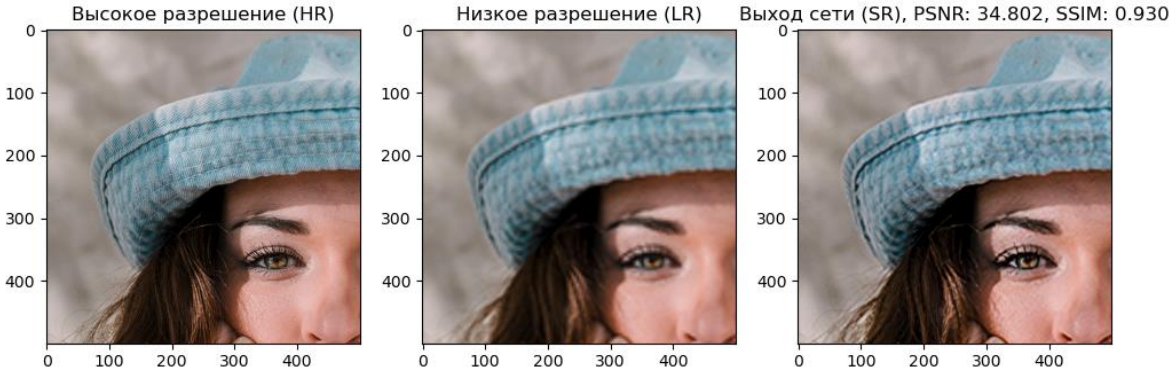


Рисунок 3.7 – Пример №2 работы сети

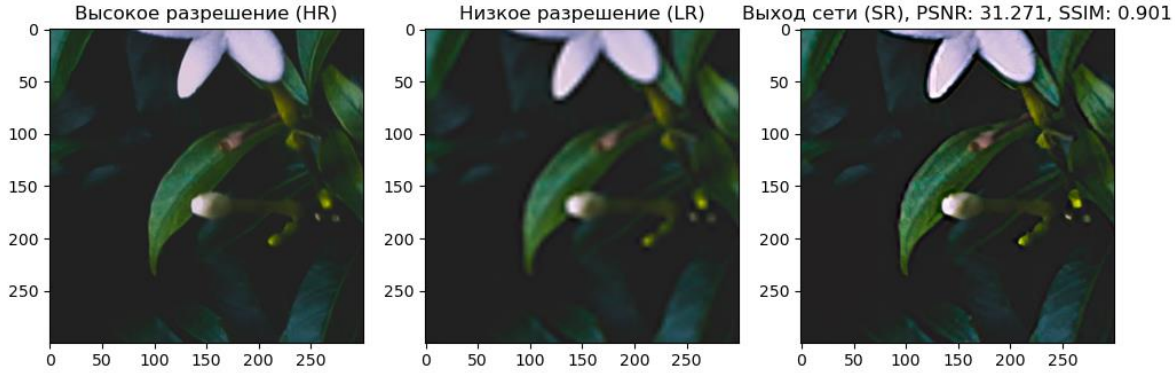


Рисунок 3.8 – Пример №3 работы сети

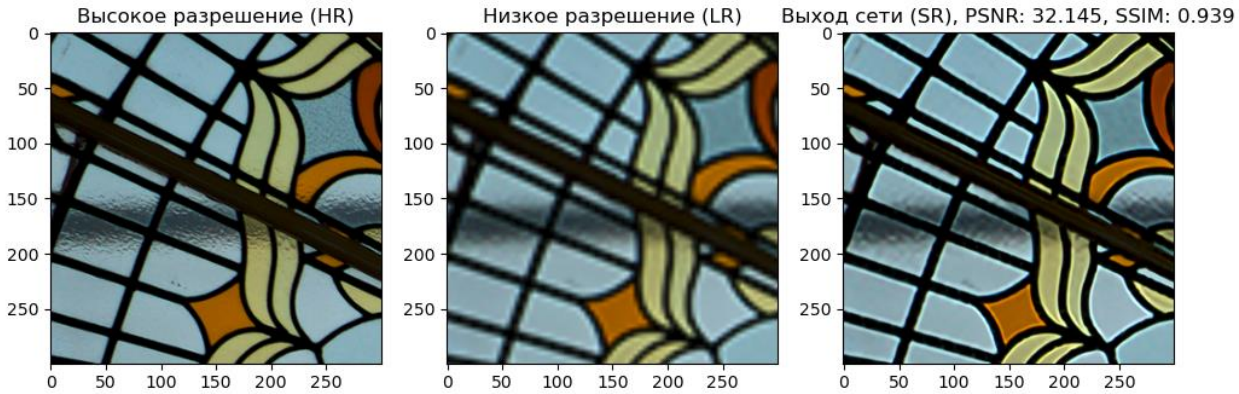


Рисунок 3.9 – Пример №4 работы сети

3.5 Сравнительный анализ

Оценка качества работы сети проводилась с помощью расчета значений метрик качества $PSNR$ и $SSIM$ (таблица 3.8).

Таблица 3.8 – Подсчет метрик качества для тестовой выборки

	Ближайший сосед	Билинейная интерполяция	Бикубическая интерполяция	Интерполяция Ланцоша	GAN
$PSNR$	29.40	29.84	30.64	30.51	32.05
$SSIM$	0.79	0.84	0.87	0.88	0.91

По результатам сравнительного анализа можно сделать вывод о том, что обученная генеративно-состязательная сеть обладает наибольшей эффективностью среди представленных методов повышения разрешения изображений.

3.6 Выводы по разделу

В данном разделе была описана архитектура используемой генеративно-состязательной нейронной сети, описаны основной алгоритм работы программы и алгоритм обучения сети.

В качестве результатов представлены графики обучения сети, на которых визуализированы изменения функций потерь для генератора и дискриминатора в процессе обучения, а также сравнительные рисунки сгенерированных изображений с изображениями низкого и высокого разрешения.

Для демонстрации эффективности работы обученной нейронной сети был проведен сравнительный анализ, где были рассчитаны метрики качества для изображений, сгенерированных нейросетью и изображений, увеличенных с помощью интерполяционных методов. Расчет значений метрик производился на тестовой выборке.

ЗАКЛЮЧЕНИЕ

Цель данной работы состояла в разработке нейронной сети для решения задачи повышения разрешения изображения без потери качества.

В соответствии с целью, в первом разделе был проведен обзор существующих методов и приложений для решения поставленной задачи. Рассмотрены их преимущества и недостатки. Также был сделан вывод о выборе эффективного метода решения – генеративно-состязательной нейронной сети.

Во втором разделе была рассмотрена математическая модель разрабатываемой нейронной сети, в том числе слои и функции активации, из которых строится архитектура модели. Для обучения сети использовался метод обратного распространения ошибки и оптимизатор Адам, а в качестве функции потерь использовалась функция потерь восприятия. Для оценки качества работы сети и сравнительного анализа использовались метрики *PSNR* и *SSIM*.

В третьем разделе была описана архитектура используемой нейронной сети, а также схемы алгоритма работы программы. В данном разделе была реализована модель нейронной сети и проверена на тестовых данных.

В результате работы, была получена обученная модель нейронной сети, решающая поставленную задачу.

В ходе работы были решены следующие задачи:

1) исследованы существующие методы для решения задачи повышения разрешения изображения без потери качества;

2) проведен сбор и подготовка данных для обучения нейронной сети;

3) разработана математическая модель искусственной нейронной сети;

4) проведено обучение сети и оценка качества ее модели на экспериментальных данных.

Таким образом, все поставленные задачи полностью решены, а цель достигнута. В дальнейшем планируется применение нейронной сети в задачах повышения разрешения видео.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Ахирвар, К. Состязательные сети. Проекты: учебное пособие / К. Ахирвар; перевод с английского В. А. Яроцкого. – Москва: ДМК Пресс, 2020. – 252 с. – ISBN 978-5-97060-783-1. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/140586 (дата обращения: 27.05.2020). – Режим доступа: для авториз. пользователей.

2 Бизли, Д. Python. Книга рецептов / Д. Бизли, Б. Джонс; пер. с англ. Б.В. Уварова. – М.: ДМК Пресс, 2019. – 648 с. – ISBN 978-5-97060-751-0. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/131723 (дата обращения: 11.01.2020). – Режим доступа: для авториз. пользователей.

3 Боровская, Е.В. Основы искусственного интеллекта: учебное пособие / Е.В. Боровская, Н.А. Давыдова. – 3-е изд. – Москва: Лаборатория знаний, 2016. – 130 с. – ISBN 978-5-00101-421-8. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/84083 (дата обращения: 14.02.2020). – Режим доступа: для авториз. пользователей.

4 Бредихин, А.И. АЛГОРИТМЫ ОБУЧЕНИЯ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ / А.И. Бредихин // Вестник Югорского государственного университета. – 2019. – № 1. – С. 41-54. – ISSN 1816-9228. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/journal/issue/310844 (дата обращения: 17.01.2020). – Режим доступа: для авториз. пользователей.

5 Вакуленко, С.А. Практический курс по нейронным сетям: учебное пособие / С.А. Вакуленко, А.А. Жихарева. – Санкт-Петербург: НИУ ИТМО, 2018. – 71 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/136500 (дата обращения: 07.01.2020). – Режим доступа: для авториз. пользователей.

6 Васильев, А.Н. Python на примерах. Практический курс по программированию: учебное пособие / А.Н. Васильев. – 3-е изд. –

Санкт-Петербург: Наука и Техника, 2019. – 432 с. – ISBN 978-5-94387-781-0. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/139151 (дата обращения: 29.04.2020). – Режим доступа: для авториз. пользователей.

7 Галушкин, А.И. Нейронные сети: основы теории / А.И. Галушкин. – Москва: Горячая линия-Телеком, 2017. – 496 с. – ISBN 978-5-9912-0082-0. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/111043 (дата обращения: 07.02.2020). – Режим доступа: для авториз. пользователей.

8 Гарсия, Г. Обработка изображений с помощью OpenCV / Г. Гарсия, О. Суарес; пер. с англ. А.А. Слинкина. – Москва: ДМК Пресс, 2016. – 210 с. – ISBN 978-5-97060-387-7. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/90116 (дата обращения: 10.12.2019). – Режим доступа: для авториз. пользователей.

9 Рамсундар, Б. Глубокое обучение в биологии и медицине / Б. Рамсундар, П. Истман, П. Уолтерс, В. Панде; перевод с английского В.С. Яценкова. – Москва: ДМК Пресс, 2020. – 202 с. – ISBN 978-5-97060-791-6. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/131725 (дата обращения: 17.01.2020). – Режим доступа: для авториз. пользователей.

10 Гонсалес, Р. Цифровая обработка изображений: учебник / Р. Гонсалес, Р. Вудс. – Москва: Техносфера, 2005. – 1072 с.

11 Гудфеллоу, Я. Глубокое обучение / Я. Гудфеллоу, И. Бенджио, А. Курвилль; пер. с англ. А.А. Слинкина. – 2-е изд. Москва: ДМК Пресс, 2018. – 652 с. – ISBN 978-5-97060-618-6. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/107901 (дата обращения: 10.12.2019). – Режим доступа: для авториз. пользователей.

12 Душкин, Р.В. Искусственный интеллект / Р.В. Душкин. – Москва: ДМК Пресс, 2019. – 280 с. – ISBN 978-5-97060-787-9. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com

/book/131703 (дата обращения: 23.01.2020). – Режим доступа: для авториз. пользователей.

13 Крутиков, В.И. Анализ данных: учебное пособие / В.И. Крутиков, В.В. Мешечкин. – Кемерово: КемГУ, 2014. – 138 с. – ISBN 978-5-8353-1770-7. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/61396 (дата обращения: 07.02.2020). – Режим доступа: для авториз. пользователей.

14 Макаренко, А.А. Практикум по цифровой обработке сигналов: учебное пособие / А.А. Макаренко. – Санкт-Петербург: НИУ ИТМО, 2014. – 50 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/71007 (дата обращения: 27.03.2020). – Режим доступа: для авториз. пользователей.

15 Надеждин, Н.Я. Введение в цифровую фотографию: учебное пособие / Н.Я. Надеждин. – 2-е изд. – Москва: ИНТУИТ, 2016. – 281 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/100675 (дата обращения: 07.05.2020). – Режим доступа: для авториз. пользователей.

16 Нуньес-Иглесиас, Х. Элегантный SciPy / Х. Нуньес-Иглесиас, В.Д. Уолт, Х. Дэшноу. – Москва: ДМК Пресс, 2018. – 266 с. – ISBN 978-5-97060-600-1. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/116124 (дата обращения: 25.03.2020). – Режим доступа: для авториз. пользователей.

17 Памперла, М. Глубокое обучение и игра в го: руководство / М. Памперла, К. Фергюсон; перевод с английского М. А. Райтмана. – Москва: ДМК Пресс, 2020. – 372 с. – ISBN 978-5-97060-769-5. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com /book/140596 (дата обращения: 30.05.2020). – Режим доступа: для авториз. пользователей.

18 Паттерсон, Д. Глубокое обучение с точки зрения практика / Д. Паттерсон, А. Гибсон. – Москва: ДМК Пресс, 2018. – 418 с. – ISBN 978-5-

97060-481-6. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/116122 (дата обращения: 11.05.2020). – Режим доступа: для авториз. пользователей.

19 Ростовцев, В.С. Искусственные нейронные сети: учебник / В.С. Ростовцев. – Санкт-Петербург: Лань, 2019. – 216 с. – ISBN 978-5-8114-3768-9. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/122180 (дата обращения: 07.01.2020). – Режим доступа: для авториз. пользователей.

20 Созыкин, А.В. ОБЗОР МЕТОДОВ ОБУЧЕНИЯ ГЛУБОКИХ НЕЙРОННЫХ СЕТЕЙ / А.В. Созыкин // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. – 2017. – № 3. – С. 28-59. – ISSN 2305-9052. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/journal/issue/306705 (дата обращения: 17.01.2020). – Режим доступа: для авториз. пользователей.

21 Сотник, С.Л. Проектирование систем искусственного интеллекта: учебное пособие / С.Л. Сотник. – 2-е изд. – Москва: ИНТУИТ, 2016. – 228 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/100395 (дата обращения: 07.02.2020). – Режим доступа: для авториз. пользователей.

22 Сузи, Р.А. Язык программирования Python: учебное пособие / Р.А. Сузи. – 2-е изд. – Москва: ИНТУИТ, 2016. – 350 с. – ISBN 5-9556-0058-2. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/100546 (дата обращения: 12.04.2020). – Режим доступа: для авториз. пользователей.

23 Танг, Д. Умные мобильные проекты с Tensorflow / Д. Танг; перевод с английского А.В. Логунова. – Москва: ДМК Пресс, 2019. – 384 с. – ISBN 978-5-97060-706-0. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/131679 (дата обращения: 28.01.2020). – Режим доступа: для авториз. пользователей.

24 Федотов, А.А. Введение в цифровую обработку биомедицинских изображений: учебное пособие / А.А. Федотов. – Санкт-Петербург: Лань, 2019. – 108 с. – ISBN 978-5-8114-3458-9. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/112697 (дата обращения: 26.02.2020). – Режим доступа: для авториз. пользователей.

25 Фисенко, В.Т. Компьютерная обработка и распознавание изображений: учебное пособие / В.Т. Фисенко, Т.Ю. Фисенко. – Санкт-Петербург: НИУ ИТМО, 2008. – 192 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/40795 (дата обращения: 17.03.2020). – Режим доступа: для авториз. пользователей.

26 Франсуа, Ш. Глубокое обучение на Python: учебник / Ш. Франсуа. – СПб.: Питер, 2018. – 400 с.:ил.

27 Шарден, Б. Крупномасштабное машинное обучение вместе с Python: учебное пособие / Б. Шарден, Л. Массарон, А. Боскетти; перевод с английского А.В. Логунова. – Москва: ДМК Пресс, 2018. – 358 с. – ISBN 978-5-97060-506-6. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/105836 (дата обращения: 27.01.2020). – Режим доступа: для авториз. пользователей.

28 Ясницкий, Л. Н. Интеллектуальные системы: учебник / Л.Н. Ясницкий. – Москва: Лаборатория знаний, 2016. – 224 с. – ISBN 978-5-00101-417-1. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/90254 (дата обращения: 31.05.2020). – Режим доступа: для авториз. пользователей.

29 Krohn J., Beyleveld G., Bassens A. Deep learning illustrated. – London: Pearson Education, 2019. – 871 p.

30 Ledig, C. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network / C. Ledig, L. Theis, F. Huszar [Электронный ресурс]. – Режим доступа: https://arxiv.org/pdf/1609.04802.pdf. – Загл. с экрана. (дата обращения: 20.10.2019).

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1
Код для подсчета метрик качества

```
import numpy as np
import cv2
import math
import os
from skimage.metrics import structural_similarity as ssim

def psnr(img1, img2):
    img1 = img1.astype(np.float64)
    img2 = img2.astype(np.float64)
    mse = np.mean((img1 - img2)**2)
    if mse == 0:
        return float('inf')
    return 20 * math.log10(255.0 / math.sqrt(mse))

def main():
    test_hr = 'test/HR'
    test_sr = 'test/SR'
    images = os.listdir(test_hr)
    PSNR = 0
    SSIM = 0

    for image in images:
      hr = cv2.imread(os.path.join(test_hr, image))
      sr = cv2.imread(os.path.join(test_sr, image))
      PSNR += psnr(sr, hr)

      gray_hr = cv2.cvtColor(hr, cv2.COLOR_BGR2GRAY)
      gray_sr = cv2.cvtColor(sr, cv2.COLOR_BGR2GRAY)
      SSIM += ssim(gray_hr, gray_sr, full=True)[0]

    print(f"SSIM: {SSIM / len(images)}")
    print(f'PSNR: {PSNR / len(images)}')


if __name__ == '__main__':
    main()
```

ПРИЛОЖЕНИЕ 2
Код программы

```
# Импорт библиотек
import os
import numpy as np
import math
import itertools
import sys

import torchvision.transforms as transforms
from torchvision.utils import save_image, make_grid

from torch.utils.data import DataLoader
from torch.autograd import Variable

from models import *
from datasets import *

import torch.nn as nn
import torch.nn.functional as F
import torch

os.makedirs("images", exist_ok=True)
os.makedirs("saved_models", exist_ok=True)

epoch = 0
n_epochs = 200
dataset_name = 'train_HR'
batch_size = 4
lr = 0.0002
b1 = 0.5
b2 = 0.999
decay_epoch = 100
n_cpu = 8
hr_height = 256
hr_width = 256
channels = 3
hr_shape = (hr_height, hr_width)
sample_interval = 2500
checkpoint_interval = 2500

cuda = torch.cuda.is_available()

# Инициализация моделей
generator = GeneratorResNet()
discriminator = Discriminator(input_shape=(channels, *hr_shape))
feature_extractor = FeatureExtractor()


# Перевод сверточной сети в режим инференса
feature_extractor.eval()

# Losses
criterion_GAN = torch.nn.MSELoss()
criterion_content = torch.nn.L1Loss()
```

```
if cuda:
    generator = generator.cuda()
    discriminator = discriminator.cuda()
    feature_extractor = feature_extractor.cuda()
    criterion_GAN = criterion_GAN.cuda()
    criterion_content = criterion_content.cuda()

if epoch != 0:

generator.load_state_dict(torch.load("saved_models/generator_%d.pth"))

discriminator.load_state_dict(torch.load("saved_models/discriminator_%
d.pth"))

# Оптимизаторы
optimizer_G = torch.optim.Adam(generator.parameters(), lr=lr,
betas=(b1, b2))
optimizer_D = torch.optim.Adam(discriminator.parameters(), lr=lr,
betas=(b1, b2))

Tensor = torch.cuda.FloatTensor if cuda else torch.Tensor

dataloader = DataLoader(
    ImageDataset(os.path.join('div2k', dataset_name),
hr_shape=hr_shape),
    batch_size=batch_size,
    shuffle=True,
    num_workers=n_cpu,
)

# ----------
#  Обучение
# ----------

for epoch in range(epoch, n_epochs):
    for i, imgs in enumerate(dataloader):

        # Формировавние входа модели
        imgs_lr = Variable(imgs["lr"].type(Tensor))
        imgs_hr = Variable(imgs["hr"].type(Tensor))

        # Adversarial ground truths
        valid = Variable(Tensor(np.ones((imgs_lr.size(0),
*discriminator.output_shape))), requires_grad=False)
        fake = Variable(Tensor(np.zeros((imgs_lr.size(0),
*discriminator.output_shape))), requires_grad=False)

        # ------------------
        #  Обучение генератора
        # ------------------

        optimizer_G.zero_grad()

        # Генерация HR изображения
        gen_hr = generator(imgs_lr)

        # Adversarial loss
```

```
        loss_GAN = criterion_GAN(discriminator(gen_hr), valid)

        # Content loss
        gen_features = feature_extractor(gen_hr)
        real_features = feature_extractor(imgs_hr)
        loss_content = criterion_content(gen_features,
real_features.detach())

        # Total loss
        loss_G = loss_content + 1e-3 * loss_GAN

        loss_G.backward()
        optimizer_G.step()

        # ----------------------
        #  Обучение дискриминатора
        # ----------------------

        optimizer_D.zero_grad()

        # Loss of real and fake images
        loss_real = criterion_GAN(discriminator(imgs_hr), valid)
        loss_fake = criterion_GAN(discriminator(gen_hr.detach()),
fake)

        # Total loss
        loss_D = (loss_real + loss_fake) / 2

        loss_D.backward()
        optimizer_D.step()

        # --------------
        #  Логирование
        # --------------

        print(f"[Epoch {epoch}/{n_epochs}] [Batch
{i}/{len(dataloader)}] [D loss: {loss_D.item()}] [G loss:
{loss_G.item()}]")

        batches_done = epoch * len(dataloader) + i
        if batches_done % sample_interval == 0:
            # Создание и сохранение сетки исходных и выходных
изображений
            imgs_lr = nn.functional.interpolate(imgs_lr,
scale_factor=4)
            gen_hr = make_grid(gen_hr, nrow=1, normalize=True)
            imgs_lr = make_grid(imgs_lr, nrow=1, normalize=True)
            img_grid = torch.cat((imgs_lr, gen_hr), -1)
            save_image(img_grid, f"images/{batches_done}.png",
normalize=False)

    if checkpoint_interval != -1 and epoch % checkpoint_interval == 0:
        # Сохранение чекопинтов модели
        torch.save(generator.state_dict(),
f"saved_models/generator_{epoch}.pth")
        torch.save(discriminator.state_dict(),
f"saved_models/discriminator_{epoch}.pth")
```

```
# Тестирование обученной сети
testloader = DataLoader(
    ImageDataset('data/test', hr_shape=hr_shape),
    batch_size=batch_size,
    shuffle=True,
    num_workers=n_cpu
)

img_lr = testdataloader[0]['lr'].type(Tensor)
gen_hr = generator(img__lr)
save_image(hen_hr, 'result_gen_hr.jpg', normalize=False)
```