**Введение:**

Целью данного исследования является разработка и обучение модели глубокого обучения для задачи суперразрешения изображений. Мы стремимся создать модель, способную повысить качество низкоразрешенных изображений до более высокого разрешения с сохранением деталей и уменьшением искажений.

**Цель работы:**

Целью данной работы является создание и обучение модели глубокого обучения для восстановления изображений высокого разрешения. Мы сосредоточимся на реализации модели Fast Super-Resolution Convolutional Neural Network (FSRCNN), которая предложена в статье [Fast Super-Resolution Convolutional Neural Networks for Image Restoration (https://arxiv.org/abs/1608.00367)](https://arxiv.org/abs/1608.00367). FSRCNN обеспечивает эффективное восстановление изображений высокого разрешения, сохраняя при этом высокую скорость обработки.

Для достижения этой цели мы выполним следующие шаги:

1. Разработаем и реализуем модель FSRCNN с использованием библиотеки TensorFlow и фреймворка Keras.
2. Подготовим и предобработаем набор данных, включая извлечение и настройку изображений высокого разрешения и их соответствующих низкоразрешенных версий.
3. Обучим модель на подготовленных данных, используя технику обучения с учителем.
4. Оценим качество модели, проведя тестирование на отдельном наборе изображений и вычислив метрики восстановления изображений, такие как PSNR (Peak Signal-to-Noise Ratio).

Данная работа направлена на создание эффективной модели восстановления изображений высокого разрешения, которая может быть полезной в различных приложениях, требующих улучшенного качества изображений. `

**Задачи:**

1. **Разработка модели:** Разработать архитектуру модели Fast Super-Resolution Convolutional Neural Network (FSRCNN) с

использованием библиотеки TensorFlow и фреймворка Keras. Это включает в себя определение слоев модели, настройку параметров и инициализацию весов.

1. **Подготовка набора данных:** Собрать и подготовить набор данных для обучения и тестирования модели. Это включает в себя загрузку изображений высокого разрешения (HR) и создание соответствующих низкоразрешенных (LR) версий с помощью бикубической интерполяции.
2. **Обучение модели:** Обучить разработанную модель на подготовленных данных. Этот шаг включает в себя выбор оптимальных гиперпараметров, таких как скорость обучения, количество эпох и размер пакета, а также оценку и управление процессом обучения.
3. **Оценка качества модели:** Оценить качество работы модели на тестовом наборе данных. Это включает в себя вычисление метрик восстановления изображений, таких как PSNR (Peak Signal-to-Noise Ratio), SSIM (Structural Similarity Index) и

визуальную оценку восстановленных изображений.

1. **Анализ результатов:** Проанализировать полученные результаты, выявить сильные и слабые стороны модели, определить области для дальнейшего улучшения и принять выводы о применимости модели в различных приложениях.

**Датасет:** [https://data.vision.ee.ethz.ch/cvl/DIV2K/ (https://data.vision.ee.ethz.ch/cvl/DIV2K/)](https://data.vision.ee.ethz.ch/cvl/DIV2K/)

`

Прежде чем передать датасет модели, необходимо выполнить несколько предварительных этапов:

1. **Понижение качества изображений:** Так как модель суперразрешения обучается на парах низкокачественных и высококачественных изображений, необходимо сначала создать низкокачественные версии высококачественных изображений. Это можно сделать с использованием метода бикубической интерполяции или других методов снижения разрешения.
2. **Разделение на train и validation части:** Датасет следует разделить на две части: часть для обучения (train) и часть для валидации (validation). Обычно train часть используется для обучения модели, в то время как validation часть используется для оценки производительности модели на данных, которые она ранее не видела.



**Далее представлено строение куросовой работы**

Downloads

├── config.yaml

├── data

* └── DIV2K\_train\_valid\_HR
* ├── 0001.png
* ├── ...
* └── 0900.png

├── train.py

├── utils

* ├── constants.py
* ├── dataset.py
* └── model.py

└── model1.h5

**Dataset**

Файл dataset представляет класс DIV2K\_Dataset , который используется для загрузки данных из набора данных DIV2K. Вот его функциональность:

1. **Инициализация**: В конструкторе класса определяются параметры, необходимые для загрузки данных, такие как путь к папке с изображениями высокого разрешения, размер пакета (batch size) и тип набора данных (train, val или test).
2. **Метод \_\_len\_\_** : Возвращает количество пакетов данных в наборе.
3. **Метод on\_epoch\_end** : Перемешивает список имен файлов с изображениями после каждой эпохи обучения.
4. **Метод \_\_getitem\_\_** : Загружает и возвращает пакет образцов изображений. Для каждого изображения высокого разрешения (HR) выполняется следующее:

Применение аугментаций (если набор данных является обучающим или валидационным).



Преобразование изображения к тензору и масштабирование значений до диапазона [0, 1].



Создание соответствующего изображения низкого разрешения (LR) путем изменения размера изображения HR методом бикубической интерполяции.



Этот класс упрощает процесс загрузки и предобработки данных перед их передачей в модель для обучения.

Ввод [14]: **import** os

**import** numpy **as** np

**import** random

**from** PIL **import** Image

**import** tensorflow **as** tf

**from** tensorflow **import** keras

**import** albumentations **as** A

**import** sys

​

**from** utils.constants **import** (

IMAGE\_FORMAT,

DOWNSAMPLE\_MODE,

COLOR\_CHANNELS,

UPSCALING\_FACTOR,

HR\_IMG\_SIZE,

LR\_IMG\_SIZE,

)

​

​

**class** DIV2K\_Dataset(keras.utils.Sequence):

"""

Загрузчик для датасета DIV2K

Ссылка на датасет: https://data.vision.ee.ethz.ch/cvl/DIV2K/

Используются только изображения высокого разрешения (HR).

Из них создаются изображения низкого разрешения (LR) с помощью бикубической интерполяции.

HR изображения для обучения: http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K\_train\_HR.zip HR изображения для валидации: http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K\_valid\_HR.zip """

**def** \_\_init\_\_(self, hr\_image\_folder: str, batch\_size: int, set\_type: str):

self.batch\_size **=** batch\_size

self.hr\_image\_folder **=** hr\_image\_folder

self.image\_fns **=** np.sort([

x **for** x **in** os.listdir(hr\_image\_folder) **if** x.endswith(IMAGE\_FORMAT)

])

**if** set\_type **==** "train":

self.image\_fns **=** self.image\_fns[:**-**200]

**elif** set\_type **==** "val":

self.image\_fns **=** self.image\_fns[**-**200:**-**100]

**else**:

self.image\_fns **=** self.image\_fns[**-**100:]

**if** set\_type **in** ["train", "val"]:

* *Преобразования изображений для обучения и валидации* self.transform **=** A.Compose(

[

A.RandomCrop(width**=**HR\_IMG\_SIZE[0], height**=**HR\_IMG\_SIZE[1], p**=**1.0), A.HorizontalFlip(p**=**0.5),

A.ColorJitter(

brightness**=**0.1, contrast**=**0.1, saturation**=**0.1, hue**=**0.1, p**=**0.8

),

]

)

**else**:

* *Преобразования изображений для тестирования* self.transform **=** A.Compose(

[

A.RandomCrop(width**=**HR\_IMG\_SIZE[0], height**=**HR\_IMG\_SIZE[1], p**=**1.0),

]

)

self.to\_float **=** A.ToFloat(max\_value**=**255)

​

**def** \_\_len\_\_(self):

**return** len(self.image\_fns) **//** self.batch\_size

​

**def** on\_epoch\_end(self):

random.shuffle(self.image\_fns)

​

**def** \_\_getitem\_\_(self, idx):

"""Возвращает пакет образцов"""

i **=** idx **\*** self.batch\_size

batch\_image\_fns **=** self.image\_fns[i : i **+** self.batch\_size]

batch\_hr\_images **=** np.zeros((self.batch\_size,) **+** HR\_IMG\_SIZE **+** (COLOR\_CHANNELS,))

batch\_lr\_images **=** np.zeros((self.batch\_size,) **+** LR\_IMG\_SIZE **+** (COLOR\_CHANNELS,))

​

**for** i, image\_fn **in** enumerate(batch\_image\_fns):

hr\_image\_pil **=** Image.open(os.path.join(self.hr\_image\_folder, image\_fn))

hr\_image **=** np.array(hr\_image\_pil)

hr\_image\_transform **=** self.transform(image**=**hr\_image)["image"] hr\_image\_transform\_pil **=** Image.fromarray(hr\_image\_transform) lr\_image\_transform\_pil **=** hr\_image\_transform\_pil.resize(

LR\_IMG\_SIZE, resample**=**DOWNSAMPLE\_MODE

)

lr\_image\_transform **=** np.array(lr\_image\_transform\_pil)

​

batch\_hr\_images[i] **=** self.to\_float(image**=**hr\_image\_transform)["image"] batch\_lr\_images[i] **=** self.to\_float(image**=**lr\_image\_transform)["image"]

​

**return** (batch\_lr\_images, batch\_hr\_images)

​



**Model**

**Этап 1: Разработка модели**

На этом этапе мы разработали модель для задачи суперразрешения изображений. Используя статью [FSRCNN](https://arxiv.org/abs/1608.00367) [(https://arxiv.org/abs/1608.00367)](https://arxiv.org/abs/1608.00367) в качестве основы, мы реализовали модель, которая состоит из нескольких сверточных слоев для извлечения признаков, сжатия, сопоставления и расширения, а также слоя деконволюции для повышения разрешения изображения.

**Методы:**

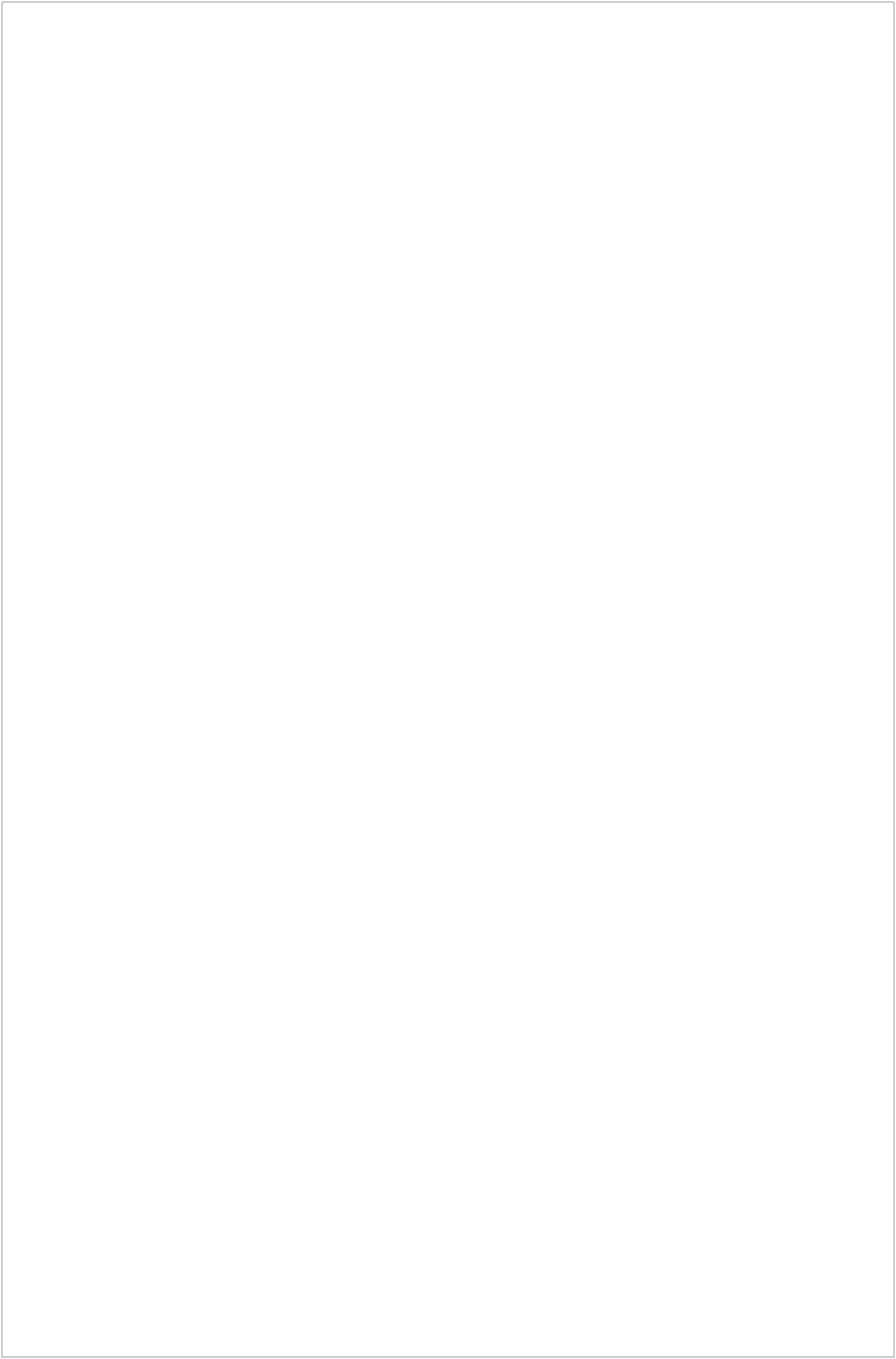
**Сверточные слои:** Мы использовали сверточные слои для извлечения признаков из входных изображений. **PReLU (Параметрическая ReLU):** Для активации сверточных слоев мы использовали PReLU, чтобы добавить нелинейность.



**Деконволюция:** Мы использовали слой деконволюции для увеличения разрешения изображения.



**Реализация:**

Ввод [15]: **from** tensorflow.keras **import** Sequential

**from** tensorflow.keras.layers **import** Conv2D, Conv2DTranspose, InputLayer, PReLU, Activation

**from** tensorflow.keras **import** initializers

​

**from** utils.constants **import** LR\_IMG\_SIZE, UPSCALING\_FACTOR, COLOR\_CHANNELS

​

​

**def** create\_model(

1. int,
2. int,
3. int,

input\_size: tuple **=** LR\_IMG\_SIZE,

upscaling\_factor: int **=** UPSCALING\_FACTOR,

color\_channels: int **=** COLOR\_CHANNELS,

):

"""

Реализация модели FSRCNN в соответствии с https://arxiv.org/abs/1608.00367

Сигмоидальная активация в выходном слое не описана в оригинальной статье,

но необходима для сопоставления прогнозов модели с исходными изображениями высокого разрешения, чтобы их значения находились в одном диапазоне [0,1]. """

model **=** Sequential()

model.add(InputLayer(input\_shape**=**(input\_size[0], input\_size[1], color\_channels)))

​

* *Извлечение признаков* model.add(

Conv2D( kernel\_size**=**5, filters**=**d, padding**=**"same",

kernel\_initializer**=**initializers.HeNormal(),

)

)

model.add(PReLU(alpha\_initializer**=**"zeros", shared\_axes**=**[1, 2]))

​

* *Сжатие*

model.add(

Conv2D( kernel\_size**=**1, filters**=**s, padding**=**"same",

kernel\_initializer**=**initializers.HeNormal(),

)

)

model.add(PReLU(alpha\_initializer**=**"zeros", shared\_axes**=**[1, 2]))

​

* *Сопоставление* **for** \_ **in** range(m):

model.add(

Conv2D( kernel\_size**=**3, filters**=**s, padding**=**"same",

kernel\_initializer**=**initializers.HeNormal(),

)

)

model.add(PReLU(alpha\_initializer**=**"zeros", shared\_axes**=**[1, 2]))

​

* *Расширение*

model.add(Conv2D(kernel\_size**=**1, filters**=**d, padding**=**"same"))

model.add(PReLU(alpha\_initializer**=**"zeros", shared\_axes**=**[1, 2]))

​

* *Деконволюция*

model.add(

Conv2DTranspose( kernel\_size**=**9, filters**=**color\_channels, strides**=**upscaling\_factor, padding**=**"same",

kernel\_initializer**=**initializers.RandomNormal(mean**=**0, stddev**=**0.001),

)

)

​

**return** model

​

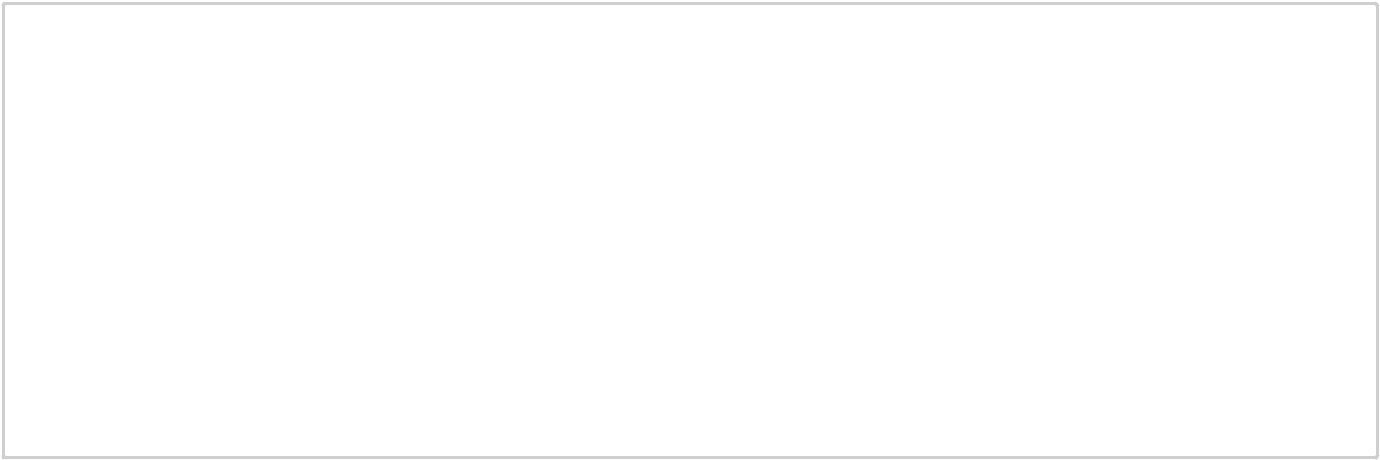
**Constants**

Файл constants содержит константы, используемые в проекте для определения формата изображений, режима сжатия, количества каналов цветности, размеров изображений высокого и низкого разрешений. Вот их краткое описание:

1. **IMAGE\_FORMAT** : Формат изображений в наборе данных. В данном случае, это .png .
2. **DOWNSAMPLE\_MODE** : Режим сжатия изображений при уменьшении размера. Здесь используется бикубическая интерполяция

(Image.BICUBIC).

1. **COLOR\_CHANNELS** : Количество каналов цветности в изображении. В RGB изображениях это обычно 3.
2. **HR\_IMG\_SIZE** : Размер изображений высокого разрешения (HR). Задается в виде кортежа (width, height) и выбирается на основе самого маленького изображения в наборе данных.
3. **UPSCALING\_FACTOR** : Множитель увеличения, определяющий во сколько раз увеличивается размер изображений при применении модели. В данном случае, это 4.
4. **LR\_IMG\_SIZE** : Размер изображений низкого разрешения (LR), рассчитанный из размера изображений высокого разрешения и множителя увеличения.



Ввод [16]: **from** PIL **import** Image

​

* *Формат изображения*

IMAGE\_FORMAT **=** ".png"

* *Режим для сжатия изображения*

DOWNSAMPLE\_MODE **=** Image.BICUBIC

* *Количество каналов цветности*

COLOR\_CHANNELS **=** 3

​

* *Размер изображения высокого разрешения*

HR\_IMG\_SIZE **=** (648, 648) *# Размер выбран на основе самого маленького изображения в наборе данных*

* *Коэффициент масштабирования*

UPSCALING\_FACTOR **=** 4

* *Размер изображения низкого разрешения*

LR\_IMG\_SIZE **=** (HR\_IMG\_SIZE[0] **//** UPSCALING\_FACTOR, HR\_IMG\_SIZE[1] **//** UPSCALING\_FACTOR)

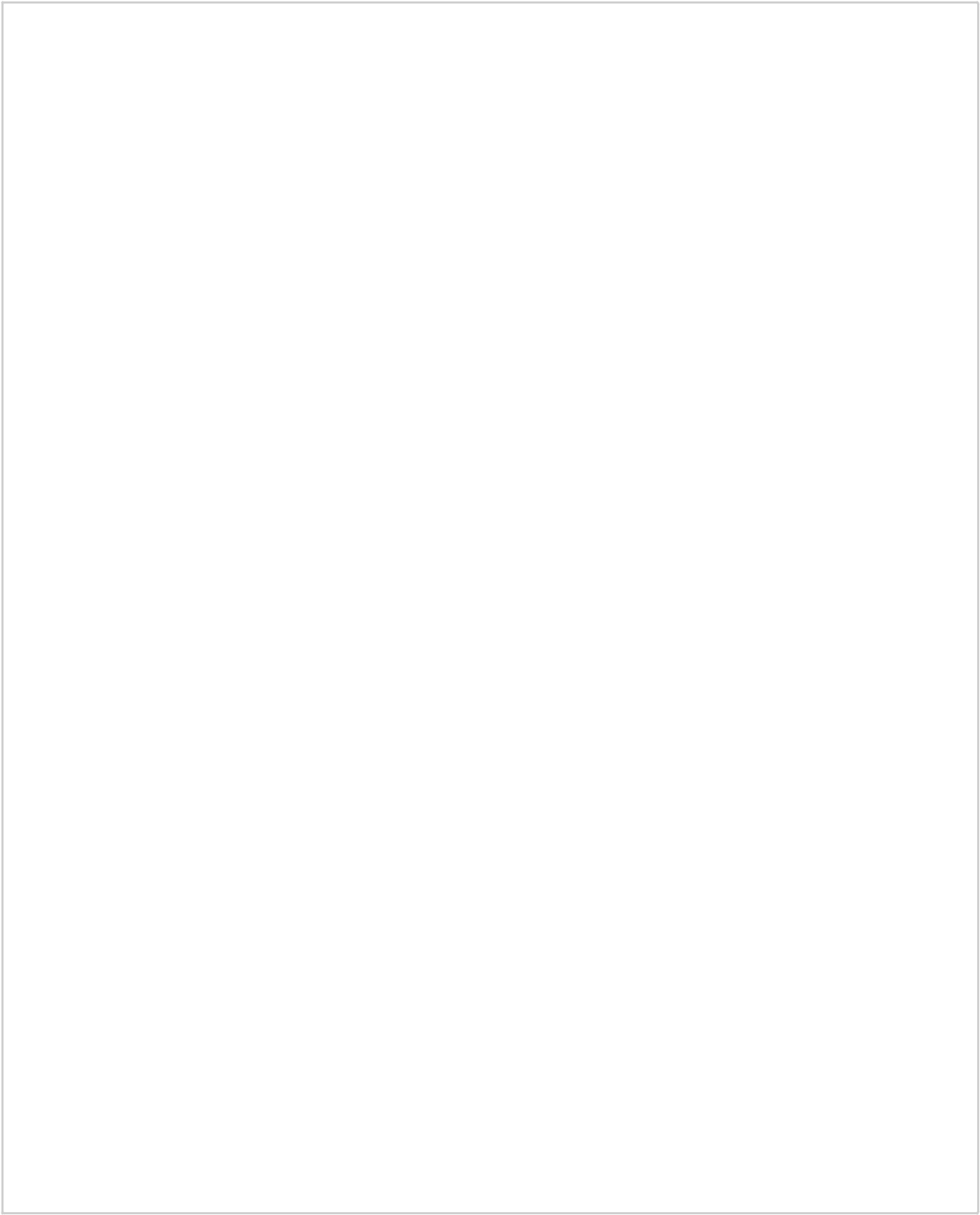
​

**Train**

Файл train содержит скрипт для обучения модели. Вот краткое описание того, что делается в этом скрипте:

1. **Загрузка конфигурации**: Считывается конфигурационный файл YAML, который содержит параметры обучения модели, такие как пути к данным, параметры модели, оптимизаторы, количество эпох и так далее.
2. **Подготовка данных**: Создаются объекты наборов данных для обучения и валидации на основе класса DIV2K\_Dataset . Для этого используются пути к папкам с изображениями высокого разрешения и параметры из конфигурационного файла, такие как размер пакета и тип набора данных.
3. **Создание модели**: Создается модель с помощью функции create\_model из модуля model.py . Параметры модели также берутся из конфигурационного файла.
4. **Компиляция модели**: Модель компилируется с оптимизатором RMSprop и функцией потерь MSE (среднеквадратичная ошибка).
5. **Настройка обратных вызовов**: Задаются обратные вызовы, такие как уменьшение скорости обучения при отсутствии улучшений, ранняя остановка и сохранение лучших весов модели.
6. **Обучение модели**: Модель обучается с использованием метода fit , передавая обучающий и валидационный наборы данных, количество эпох и другие параметры.

Этот скрипт предоставляет основной рабочий процесс для обучения модели и может быть запущен из командной строки, принимая путь к конфигурационному файлу в качестве аргумента.

Ввод [17]: **import** argparse

**import** yaml

**import** tensorflow **as** tf

**from** tensorflow **import** keras

​

**from** utils.dataset **import** DIV2K\_Dataset

**from** utils.model **import** create\_model

**from** utils.constants **import** HR\_IMG\_SIZE, DOWNSAMPLE\_MODE

​

**def** train(config\_fn: str) **-> None**:

* *Загрузка конфигурационного файла*

**with** open(config\_fn, 'r') **as** stream: config **=** yaml.safe\_load(stream)

​

* *Загрузка обучающего и валидационного наборов данных* train\_dataset **=** DIV2K\_Dataset(

hr\_image\_folder**=**config["data\_path"], batch\_size**=**config["batch\_size"], set\_type**=**"train",

)

val\_dataset **=** DIV2K\_Dataset( hr\_image\_folder**=**config["data\_path"], batch\_size**=**config["val\_batch\_size"], set\_type**=**"val",

)

​

* *Создание модели*

model **=** create\_model(d**=**config["model\_d"], s**=**config["model\_s"], m**=**config["model\_m"])

model.compile(

optimizer**=**keras.optimizers.RMSprop(learning\_rate**=**config["lr\_init"]), loss**=**"mean\_squared\_error",

)

​

* *Коллбэки для управления процессом обучения* reduce\_lr **=** keras.callbacks.ReduceLROnPlateau(

monitor**=**"loss", factor**=**0.5, patience**=**20, min\_lr**=**10e-6, verbose**=**1

)

early\_stopping **=** keras.callbacks.EarlyStopping( monitor**=**"val\_loss",

min\_delta**=**10e-6, patience**=**40, verbose**=**0, restore\_best\_weights**=True**,

)

save **=** keras.callbacks.ModelCheckpoint( filepath**=**config["weights\_fn"], monitor**=**"loss", save\_best\_only**=True**, save\_weights\_only**=False**, save\_freq**=**"epoch",

)

​

* *Обучение модели* history **=** model.fit( train\_dataset,

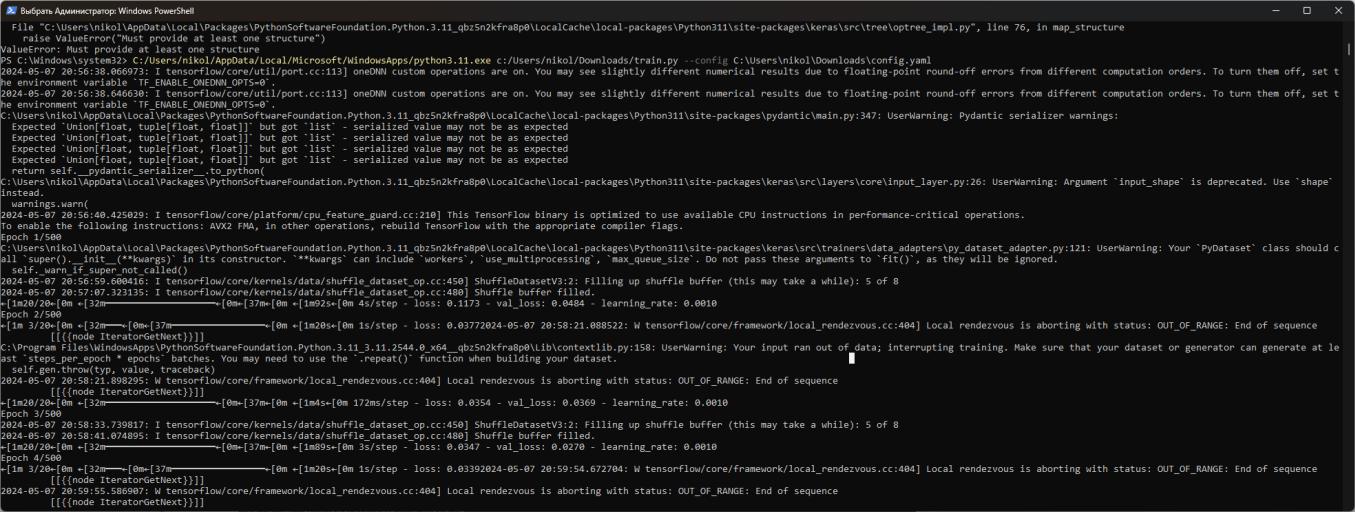
epochs**=**config["epochs"],

steps\_per\_epoch**=**config["steps\_per\_epoch"], callbacks**=**[reduce\_lr, early\_stopping, save], validation\_data**=**val\_dataset, validation\_steps**=**config["validation\_steps"],

)

​

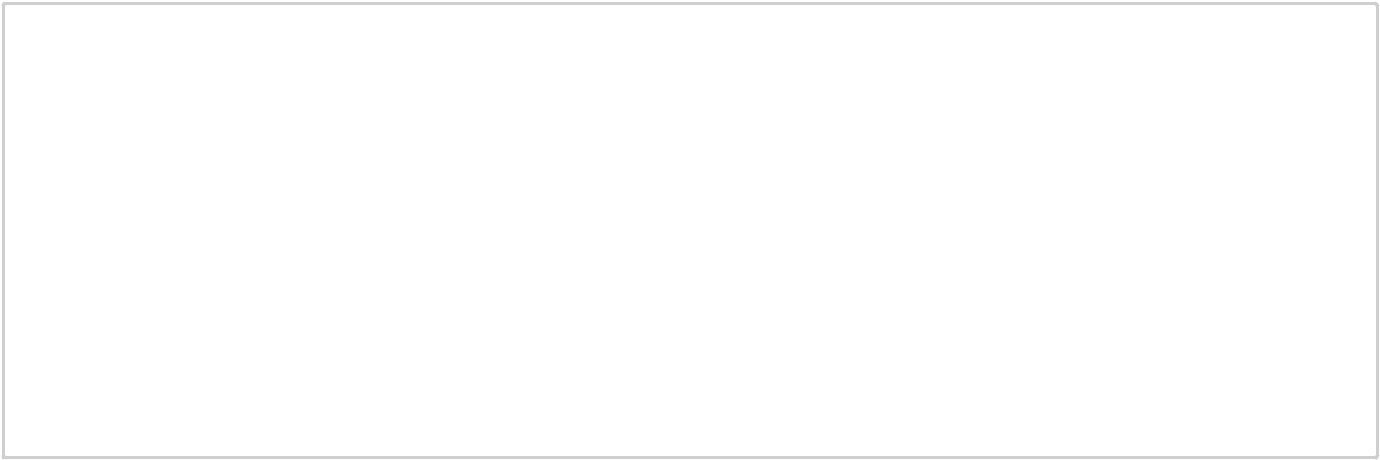
**На данном фото показан процесс обучения модели FSRCNN.**



**Config**

Файл config.yaml содержит параметры, используемые при обучении модели суперразрешения изображений на наборе данных DIV2K. Вот краткое описание каждого параметра:

1. data\_path : Путь к папке с изображениями высокого разрешения (HR), используемыми для обучения и валидации модели.
2. model\_d : Количество фильтров в сверточных слоях для извлечения признаков в модели FSRCNN.
3. model\_s : Количество фильтров в сверточных слоях для сжатия в модели FSRCNN.
4. model\_m : Количество сверточных слоев для сопоставления в модели FSRCNN.
5. lr\_init : Начальная скорость обучения (learning rate) для оптимизатора RMSprop.
6. epochs : Количество эпох обучения.
7. batch\_size : Размер пакета (batch size) образцов данных для каждой итерации обучения.
8. steps\_per\_epoch : Количество шагов (steps) обучения в каждой эпохе. Это определяет, сколько пакетов образцов будет обрабатываться за одну эпоху.
9. val\_batch\_size : Размер пакета образцов данных для каждой итерации валидации.
10. validation\_steps : Количество шагов валидации в каждой эпохе. Это определяет, сколько пакетов образцов будет обрабатываться при валидации за одну эпоху.
11. weights\_fn : Путь и формат имени файла для сохранения весов модели после каждой эпохи обучения. В данном случае используется формат, включающий номер эпохи в имени файла.



Ввод [ ]: *# %load config.yaml*

data\_path: C:\Users\nikol\Downloads\data\DIV2K\_train\_valid\_HR\

​

model\_d: 56

model\_s: 12

model\_m: 4

​

lr\_init: 0.001

epochs: 500

batch\_size: 30

steps\_per\_epoch: 20

val\_batch\_size: 20

validation\_steps: 4

​

weights\_fn: "weights/model\_{epoch:05d}.keras"

​

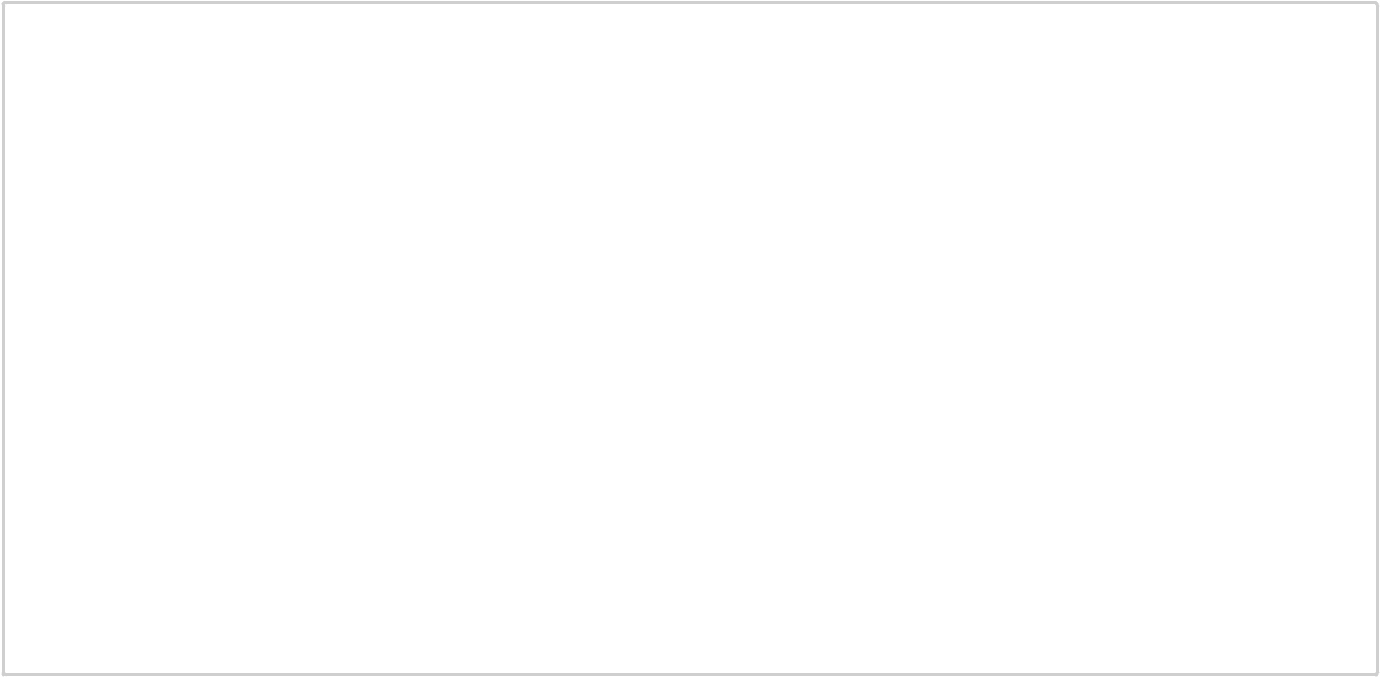
**Далее надо загрузить все файлы в один notebook, и вывести пример работы**



Ввод [1]: **%**load\_ext autoreload

**%**autoreload 2

​

Ввод [3]: **import** tensorflow **as** tf

**from** tensorflow **import** keras

**import** os

**import** numpy **as** np

**from** PIL **import** Image

**import** matplotlib.pyplot **as** plt

**import** sys

**import** yaml

**import** warnings

warnings.simplefilter(action**=**'ignore')

​

​

* *Добавляем путь к каталогу utils в список путей поиска модулей Python* sys.path.append("C:\\Users\\nikol\\Downloads\\utils")

​

* *Импортируем класс DIV2K\_Dataset из модуля dataset в каталоге utils* **from** utils.dataset **import** DIV2K\_Dataset

​

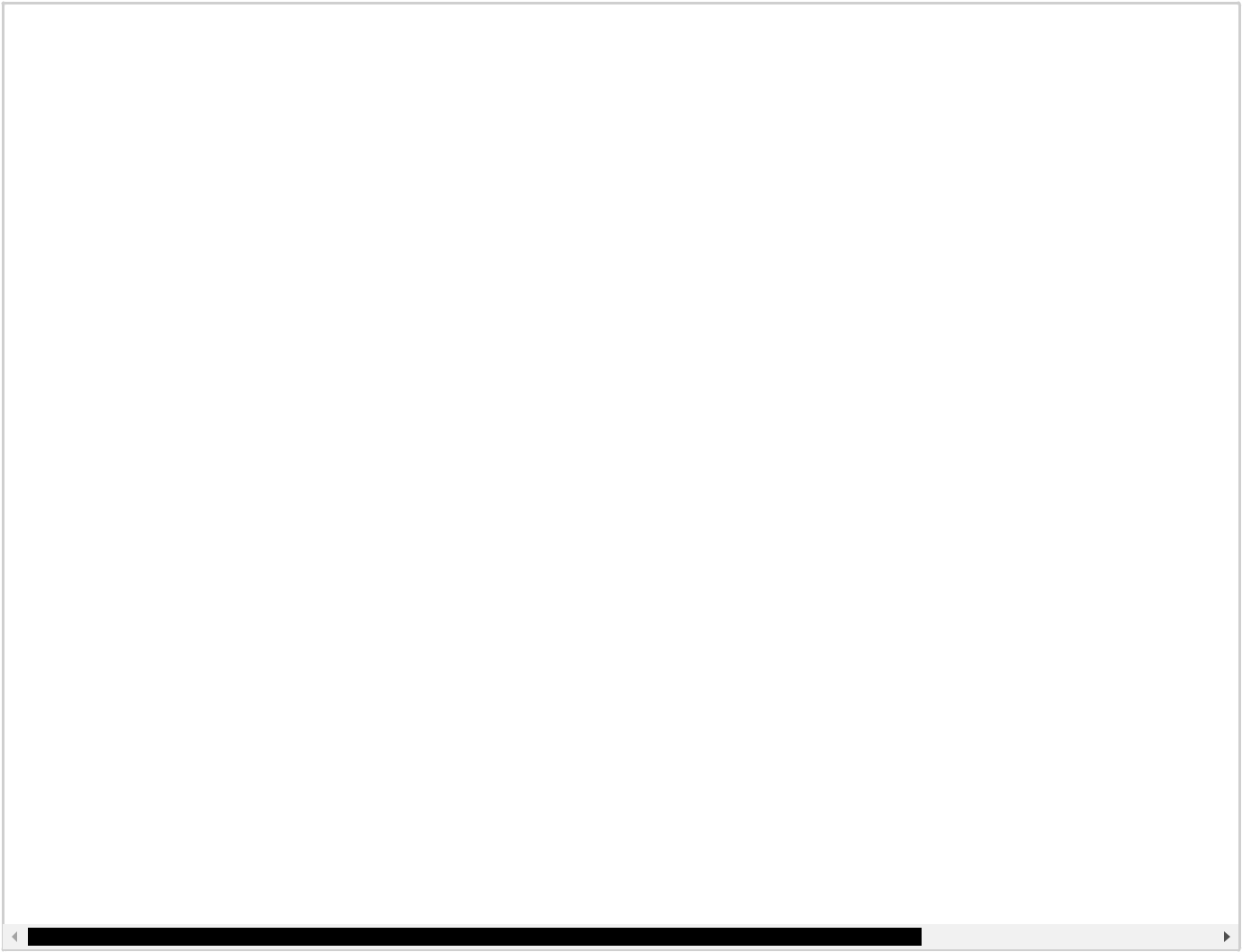
* *Импортируем функцию create\_model из модуля model в каталоге utils* **from** utils.model **import** create\_model

​

* *Импортируем константы HR\_IMG\_SIZE и DOWNSAMPLE\_MODE из модуля constants в каталоге utils* **from** utils.constants **import** HR\_IMG\_SIZE, DOWNSAMPLE\_MODE

​

**Parameters**



Ввод [4]: **import** tensorflow **as** tf

**from** tensorflow.keras.layers **import** InputLayer, Conv2D, PReLU, Conv2DTranspose **from** tensorflow.keras **import** initializers ​

* *Создание пустой модели* model **=** tf.keras.Sequential()

​

* *Добавление слоя входных данных* model.add(InputLayer(input\_shape**=**(**None**, **None**, 3)))

​

* *Добавление слоя свертки (feature extraction)*

model.add(Conv2D(kernel\_size**=**5, filters**=**56, padding**=**"same", kernel\_initializer**=**initializers.HeNormal()))

model.add(PReLU(alpha\_initializer**=**"zeros", shared\_axes**=**[1, 2]))

​

* *Добавление слоя свертки (shrinking)*

model.add(Conv2D(kernel\_size**=**1, filters**=**12, padding**=**"same", kernel\_initializer**=**initializers.HeNormal()))

model.add(PReLU(alpha\_initializer**=**"zeros", shared\_axes**=**[1, 2]))

​

* *Добавление слоев mapping* **for** \_ **in** range(4):

model.add(Conv2D(kernel\_size**=**3, filters**=**12, padding**=**"same", kernel\_initializer**=**initializers.HeNormal())) model.add(PReLU(alpha\_initializer**=**"zeros", shared\_axes**=**[1, 2]))

​

* *Добавление слоя свертки (expanding)*

model.add(Conv2D(kernel\_size**=**1, filters**=**56, padding**=**"same"))

model.add(PReLU(alpha\_initializer**=**"zeros", shared\_axes**=**[1, 2]))

​

* *Добавление слоя деконволюции (deconvolution)*

model.add(Conv2DTranspose(kernel\_size**=**9, filters**=**3, strides**=**4, padding**=**"same", kernel\_initializer**=**initializers. ​

* *Вывод структуры модели* model.summary()

​

* *Попытка загрузки весов*

model.load\_weights("C:/Users/nikol/Downloads/model1.h5")

​

​

**Model: "sequential"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩

│ conv2d (Conv2D) │ (None, None, None, 56) │ 4,256 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ p\_re\_lu (PReLU) │ (None, None, None, 56) │ 56 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_1 (Conv2D) │ (None, None, None, 12) │ 684 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ p\_re\_lu\_1 (PReLU) │ (None, None, None, 12) │ 12 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_2 (Conv2D) │ (None, None, None, 12) │ 1,308 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ p\_re\_lu\_2 (PReLU) │ (None, None, None, 12) │ 12 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_3 (Conv2D) │ (None, None, None, 12) │ 1,308 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ p\_re\_lu\_3 (PReLU) │ (None, None, None, 12) │ 12 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_4 (Conv2D) │ (None, None, None, 12) │ 1,308 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ p\_re\_lu\_4 (PReLU) │ (None, None, None, 12) │ 12 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_5 (Conv2D) │ (None, None, None, 12) │ 1,308 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ p\_re\_lu\_5 (PReLU) │ (None, None, None, 12) │ 12 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_6 (Conv2D) │ (None, None, None, 56) │ 728 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ p\_re\_lu\_6 (PReLU) │ (None, None, None, 56) │ 56 │ ├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_transpose (Conv2DTranspose) │ (None, None, None, 3) │ 13,611 │ └──────────────────────────────────────┴─────────────────────────────┴─────────────────┘

**Total params:** 24,683 (96.42 KB)

**Trainable params:** 24,683 (96.42 KB)

**Non-trainable params:** 0 (0.00 B)

Этот ячейка код создает модель нейронной сети для восстановления изображений (Restoration Model). Вот что он делает по шагам:

1. Импортируются необходимые модули TensorFlow и его компоненты для создания и обучения моделей нейронных сетей.
2. Создается пустая модель Sequential , которая будет содержать все слои нашей нейронной сети.
3. Добавляется входной слой InputLayer , который указывает форму входных данных. Здесь мы ожидаем изображения с тремя каналами цвета.
4. Добавляются последовательно слои свертки ( Conv2D ) и активации ( PReLU ). Эти слои выполняют извлечение признаков из изображения, сжатие, сопоставление и расширение.
5. Затем добавляется слой деконволюции ( Conv2DTranspose ), который выполняет обратную операцию свертки для восстановления изображения. Этот слой увеличивает размер изображения в четыре раза.
6. Метод summary() выводит структуру созданной модели, показывая количество параметров и форму выходных данных на каждом слое.
7. Попытка загрузить веса из предварительно сохраненного файла "C:/Users/nikol/Downloads/model1.h5" . Это позволяет использовать предварительно обученные веса для модели, если они доступны.

Этот код может быть частью процесса создания, обучения и использования модели для восстановления изображений, например, в приложениях по обработке изображений, сжатии изображений или улучшении качества изображений.

**Далее загружаем веса для визуализации**



Ввод [5]: *# Открытие файла config.yaml для чтения с использованием конструкции контекстного менеджера* **with** open("C:/Users/nikol/Downloads/config.yaml", "r") **as** f:

* *Загрузка данных из файла YAML в формате словаря Python* config **=** yaml.safe\_load(f)

​

* *Загрузка весов модели из файла model1.h5*

model.load\_weights("C:/Users/nikol/Downloads/model1.h5")

​

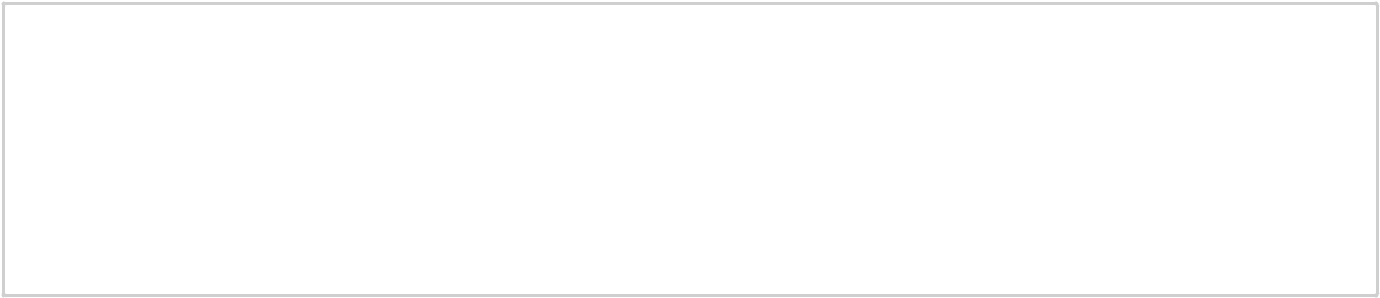
**Оценка качества**

**Данный код позволяет оценить качество модели на тестовом наборе данных путем вычисления среднего значения PSNR.**

PSNR (Peak Signal-to-Noise Ratio) - это метрика, используемая для оценки качества восстановленных изображений или видео. Она измеряет отношение максимально возможного значения сигнала к уровню шума, который присутствует в изображении. В вычислительной фотографии PSNR обычно выражается в децибелах (dB).



**Test Set**



Ввод [6]: *# Создание тестового набора данных DIV2K\_Dataset*

* *hr\_image\_folder: путь к папке с изображениями высокого разрешения (HR)*
* *batch\_size: размер пакета данных, используемый при загрузке*
* *set\_type: тип набора данных, в данном случае "test"*

test\_dataset **=** DIV2K\_Dataset(

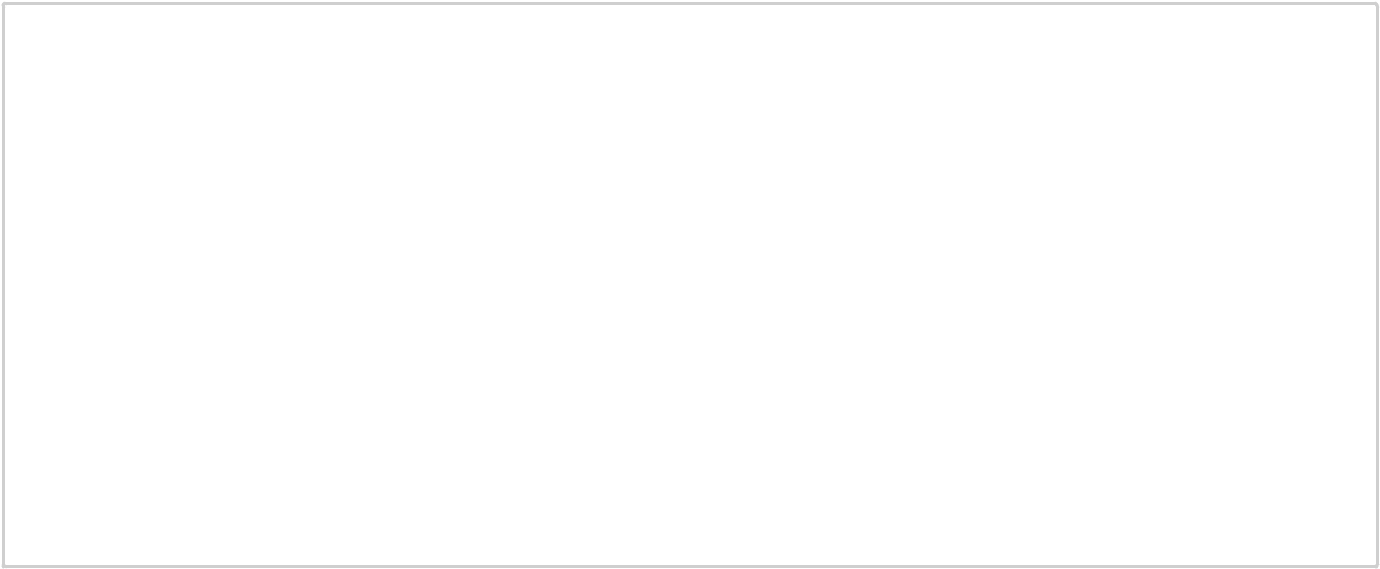
hr\_image\_folder**=**"C:/Users/nikol/Downloads/data/DIV2K\_train\_valid\_HR/",

batch\_size**=**config["val\_batch\_size"],

set\_type**=**"test",

)

​

Ввод [ ]: *# Количество запусков для вычисления среднего значения PSNR* n\_runs **=** 5

* *Список для сохранения значений PSNR* psnrs **=** []

​

* *Цикл для каждого запуска*

**for** \_ **in** range(n\_runs):

* *Цикл по пакетам данных в тестовом наборе* **for** batch **in** test\_dataset:
  + *Получение прогнозов модели для входных данных пакета* preds **=** model.predict(batch[0])
  + *Вычисление значения PSNR между истинными и предсказанными изображениями* psnr **=** tf.image.psnr(batch[1], preds, max\_val**=**1.0)
  + *Преобразование значения PSNR в список и добавление к списку psnrs*

psnr **=** psnr.numpy().tolist()

psnrs.extend(psnr)

​

* *Вывод среднего значения PSNR по всем запускам* print("Mean PSNR: {:.3f}".format(np.mean(psnrs)))

​

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 305ms/step |  |  |
|  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 285ms/step |  |  |
|  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 280ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 284ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 287ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 295ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 313ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 274ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 277ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 283ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 279ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 300ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 281ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 283ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 286ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 287ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 283ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 282ms/step |  |  |
| **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** | | 279ms/step |  |  |
| **1/1** | **0s** | 296 / t |  |  |



**Визуализация**

**Test Set**

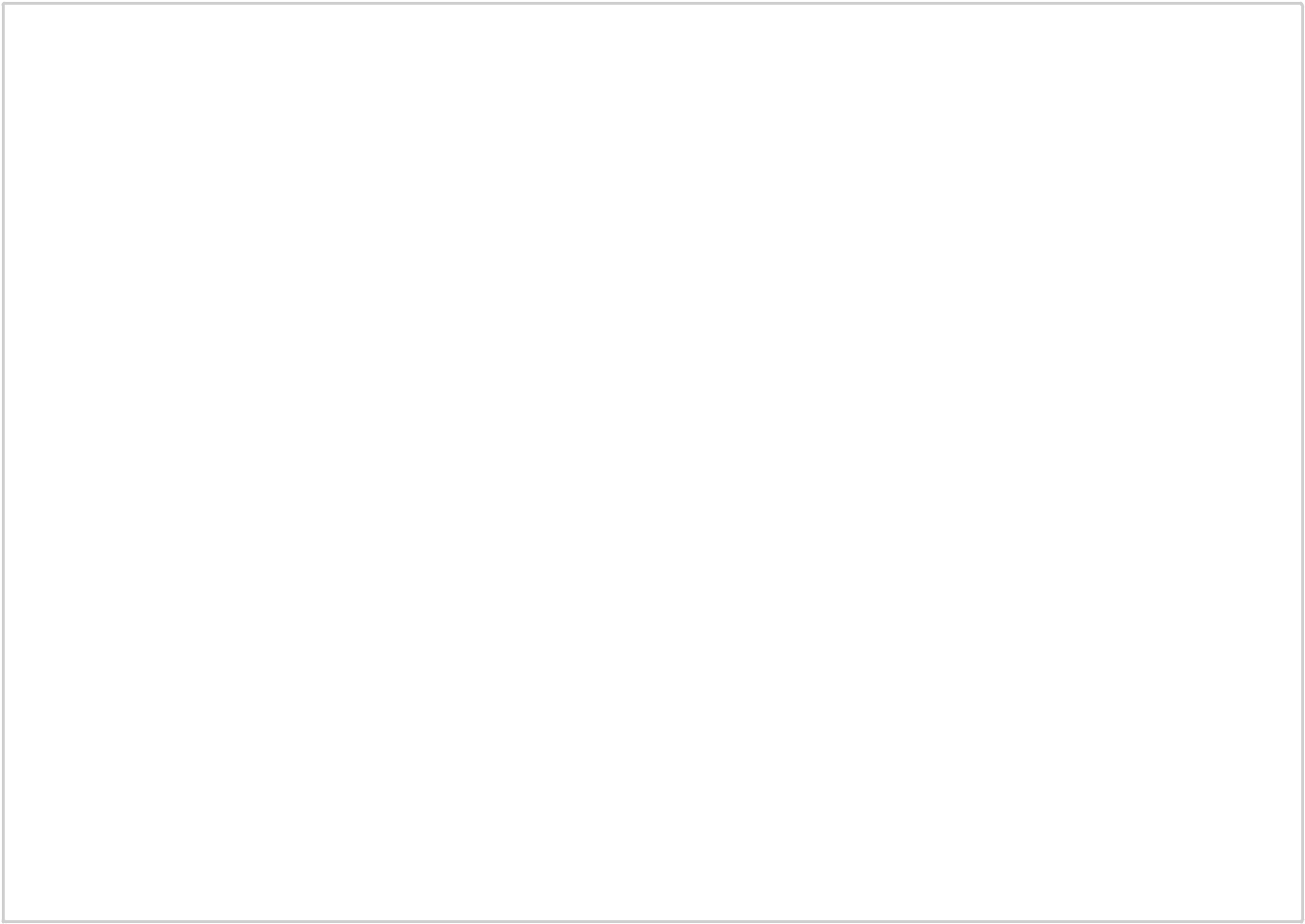


Ввод [7]: batch\_id **=** 0

* *Получение пакета данных из тестового набора* batch **=** test\_dataset.\_\_getitem\_\_(batch\_id)
* *Получение прогнозов модели на пакете данных* preds **=** model.predict(batch[0])

​

**1/1** ━━━━━━━━━━━━━━━━━━━━ **1s** 523ms/step

Ввод [8]: img\_id **=** 1

​

* *Создание фигуры для отображения изображений* plt.figure(figsize**=**[15, 15])

​

* *Отображение низкокачественного изображения* plt.subplot(2, 2, 1) plt.imshow(batch[0][img\_id]) plt.axis("off")

plt.title("LR Image")

​

* *Отображение высококачественного (оригинального) изображения* plt.subplot(2, 2, 2)

plt.imshow(batch[1][img\_id]) plt.axis("off") plt.title("HR Image")

​

* *Отображение восстановленного изображения моделью* plt.subplot(2, 2, 3) plt.imshow(preds[img\_id])

plt.axis("off")

plt.title("Restored Image")

​

* *Отображение изображения, увеличенного бикубической интерполяцией* plt.subplot(2, 2, 4)

lr\_image **=** Image.fromarray(np.array(batch[0][img\_id] **\*** 255, dtype**=**"uint8")) lr\_image\_resized **=** lr\_image.resize(HR\_IMG\_SIZE, resample**=**DOWNSAMPLE\_MODE) plt.imshow(lr\_image\_resized)

plt.axis("off")

plt.title("Bilinear Upsampling")

​

* *Автоматическое распределение макета*

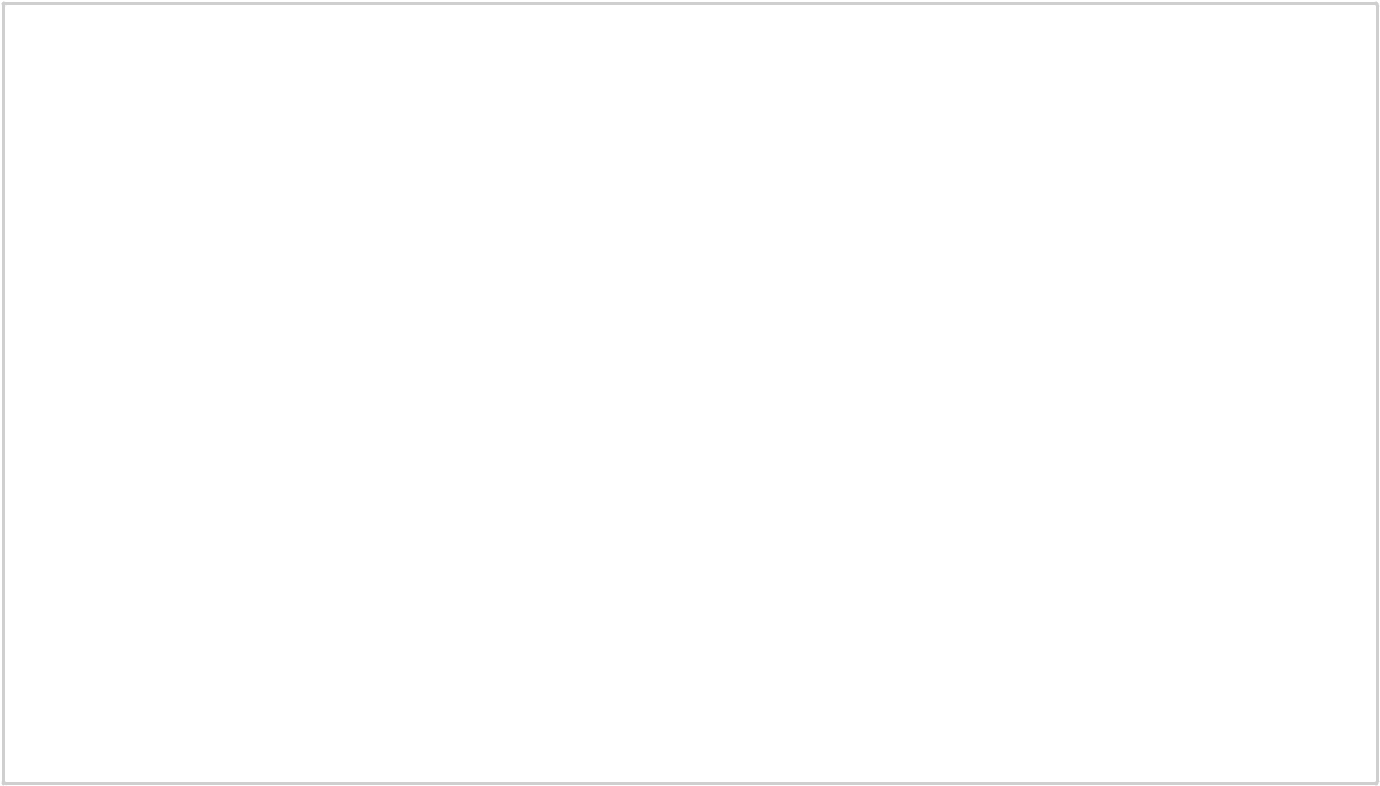
plt.tight\_layout()

plt.show()

​

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



Ввод [9]: img\_id **=** 9

​

plt.figure(figsize**=**[15, 15])

plt.subplot(2, 2, 1)

plt.imshow(batch[0][img\_id])

plt.axis("off")

plt.title("LR Image")

​

plt.subplot(2, 2, 2)

plt.imshow(batch[1][img\_id])

plt.axis("off")

plt.title("HR Image")

​

plt.subplot(2, 2, 3)

plt.imshow(preds[img\_id])

plt.axis("off")

plt.title("Restored Image")

​

​

plt.subplot(2, 2, 4)

lr\_image **=** Image.fromarray(np.array(batch[0][img\_id] **\*** 255, dtype**=**"uint8"))

lr\_image\_resized **=** lr\_image.resize(HR\_IMG\_SIZE, resample**=**DOWNSAMPLE\_MODE)

plt.imshow(lr\_image\_resized)

plt.axis("off")

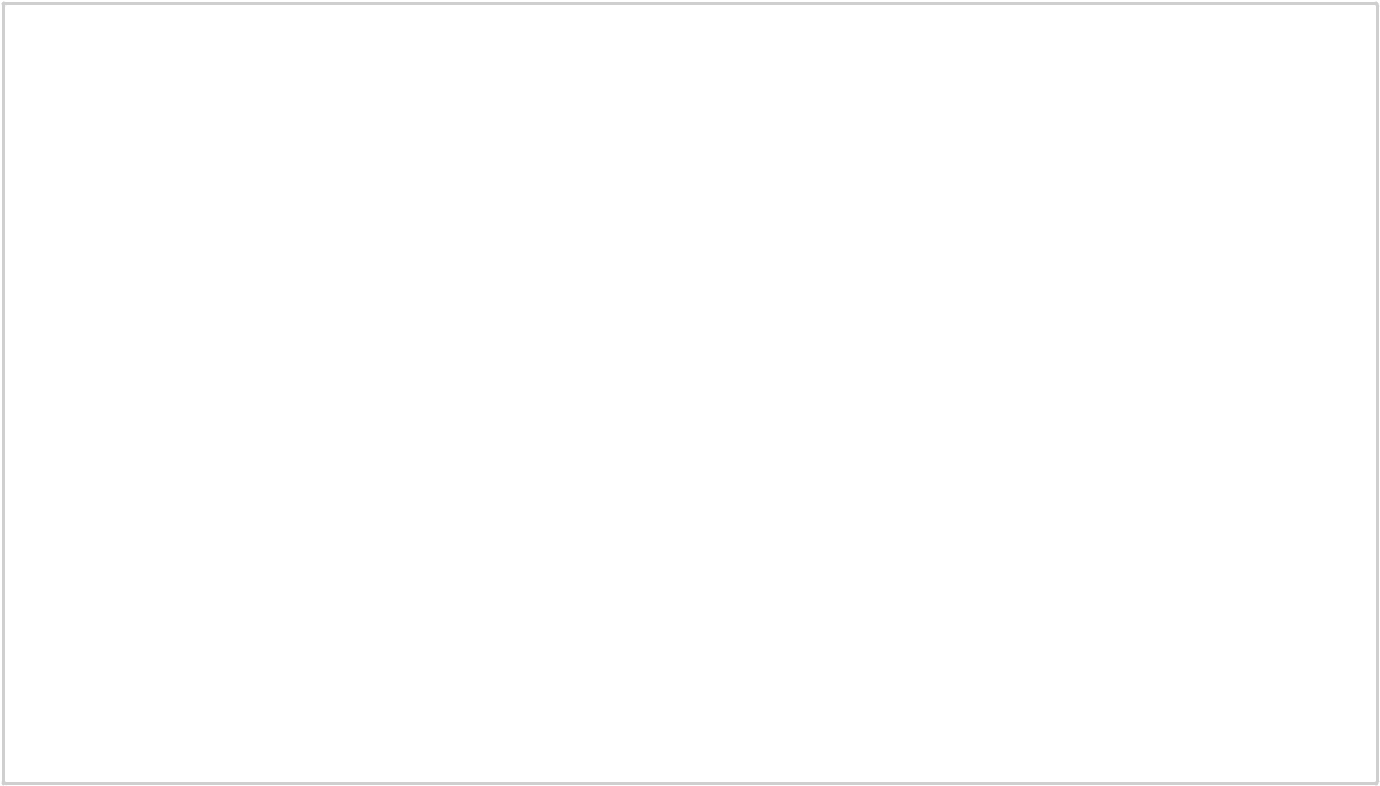
plt.title("Bilinear Upsampling")

​

plt.tight\_layout()

plt.show()



Ввод [10]: img\_id **=** 3

​

plt.figure(figsize**=**[15, 15])

plt.subplot(2, 2, 1)

plt.imshow(batch[0][img\_id])

plt.axis("off")

plt.title("LR Image")

​

plt.subplot(2, 2, 2)

plt.imshow(batch[1][img\_id])

plt.axis("off")

plt.title("HR Image")

​

plt.subplot(2, 2, 3)

plt.imshow(preds[img\_id])

plt.axis("off")

plt.title("Restored Image")

​

​

plt.subplot(2, 2, 4)

lr\_image **=** Image.fromarray(np.array(batch[0][img\_id] **\*** 255, dtype**=**"uint8"))

lr\_image\_resized **=** lr\_image.resize(HR\_IMG\_SIZE, resample**=**DOWNSAMPLE\_MODE)

plt.imshow(lr\_image\_resized)

plt.axis("off")

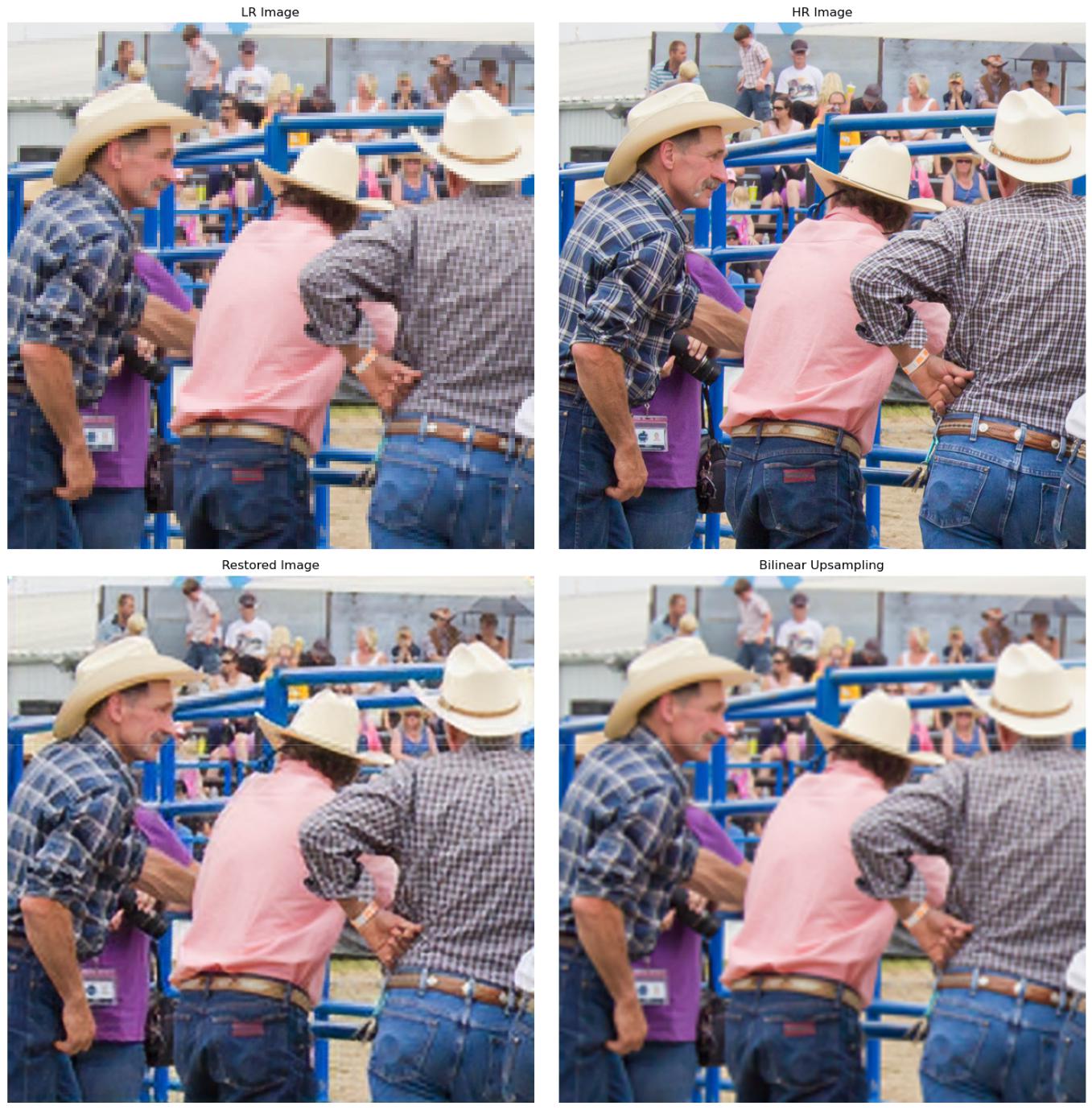
plt.title("Bilinear Upsampling")

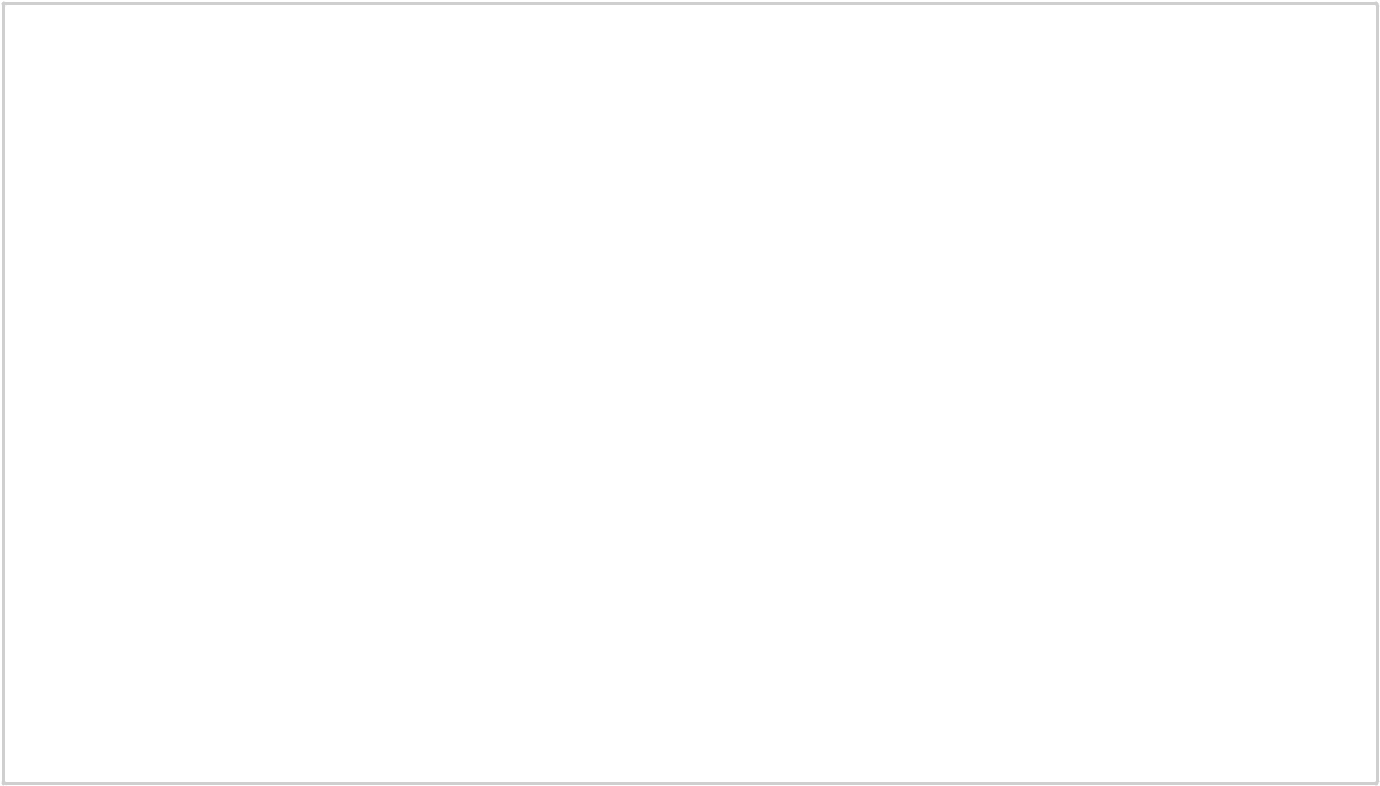
​

plt.tight\_layout()

plt.show()

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



Ввод [11]: img\_id **=** 7

​

plt.figure(figsize**=**[15, 15])

plt.subplot(2, 2, 1)

plt.imshow(batch[0][img\_id])

plt.axis("off")

plt.title("LR Image")

​

plt.subplot(2, 2, 2)

plt.imshow(batch[1][img\_id])

plt.axis("off")

plt.title("HR Image")

​

plt.subplot(2, 2, 3)

plt.imshow(preds[img\_id])

plt.axis("off")

plt.title("Restored Image")

​

​

plt.subplot(2, 2, 4)

lr\_image **=** Image.fromarray(np.array(batch[0][img\_id] **\*** 255, dtype**=**"uint8"))

lr\_image\_resized **=** lr\_image.resize(HR\_IMG\_SIZE, resample**=**DOWNSAMPLE\_MODE)

plt.imshow(lr\_image\_resized)

plt.axis("off")

plt.title("Bilinear Upsampling")

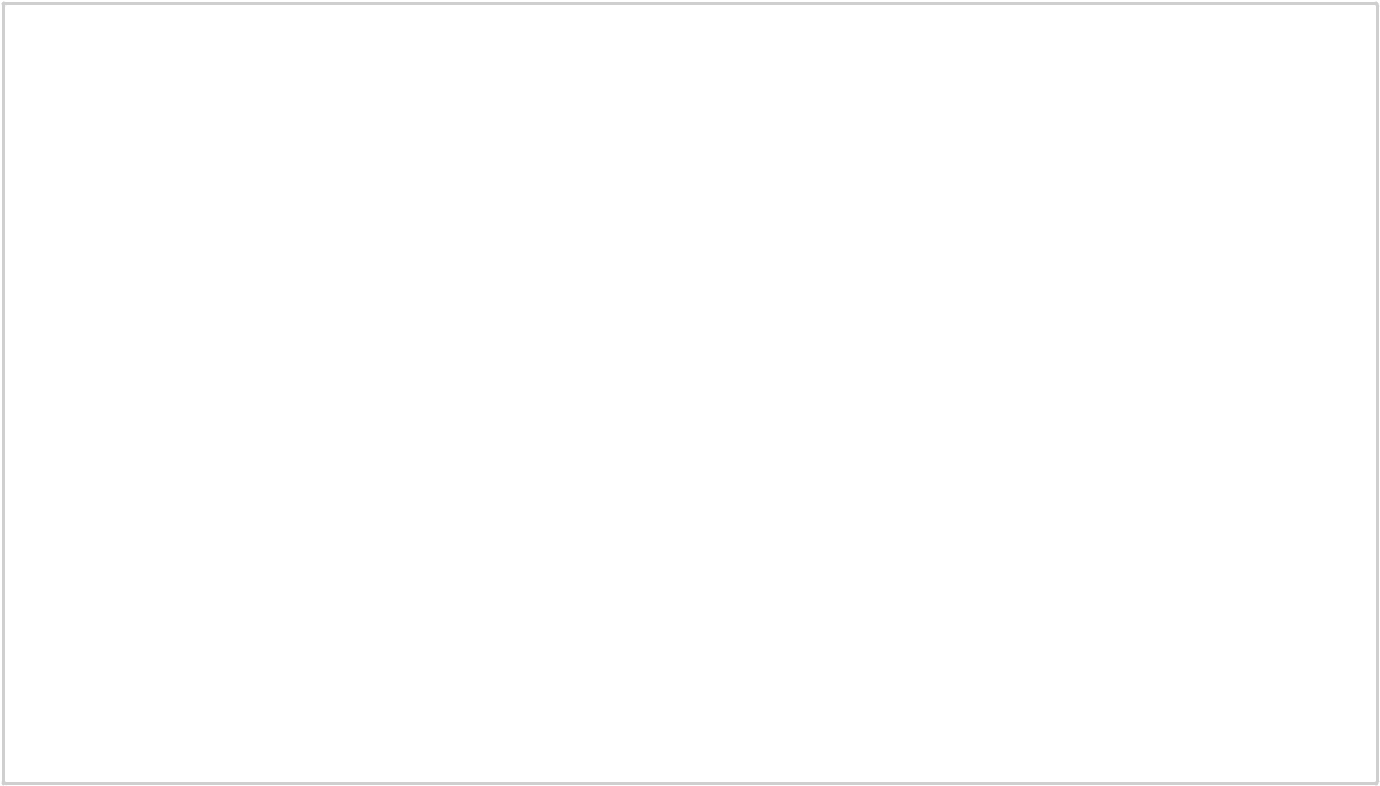
​

plt.tight\_layout()

plt.show()

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



Ввод [12]: img\_id **=** 5

​

plt.figure(figsize**=**[15, 15])

plt.subplot(2, 2, 1)

plt.imshow(batch[0][img\_id])

plt.axis("off")

plt.title("LR Image")

​

plt.subplot(2, 2, 2)

plt.imshow(batch[1][img\_id])

plt.axis("off")

plt.title("HR Image")

​

plt.subplot(2, 2, 3)

plt.imshow(preds[img\_id])

plt.axis("off")

plt.title("Restored Image")

​

​

plt.subplot(2, 2, 4)

lr\_image **=** Image.fromarray(np.array(batch[0][img\_id] **\*** 255, dtype**=**"uint8"))

lr\_image\_resized **=** lr\_image.resize(HR\_IMG\_SIZE, resample**=**DOWNSAMPLE\_MODE)

plt.imshow(lr\_image\_resized)

plt.axis("off")

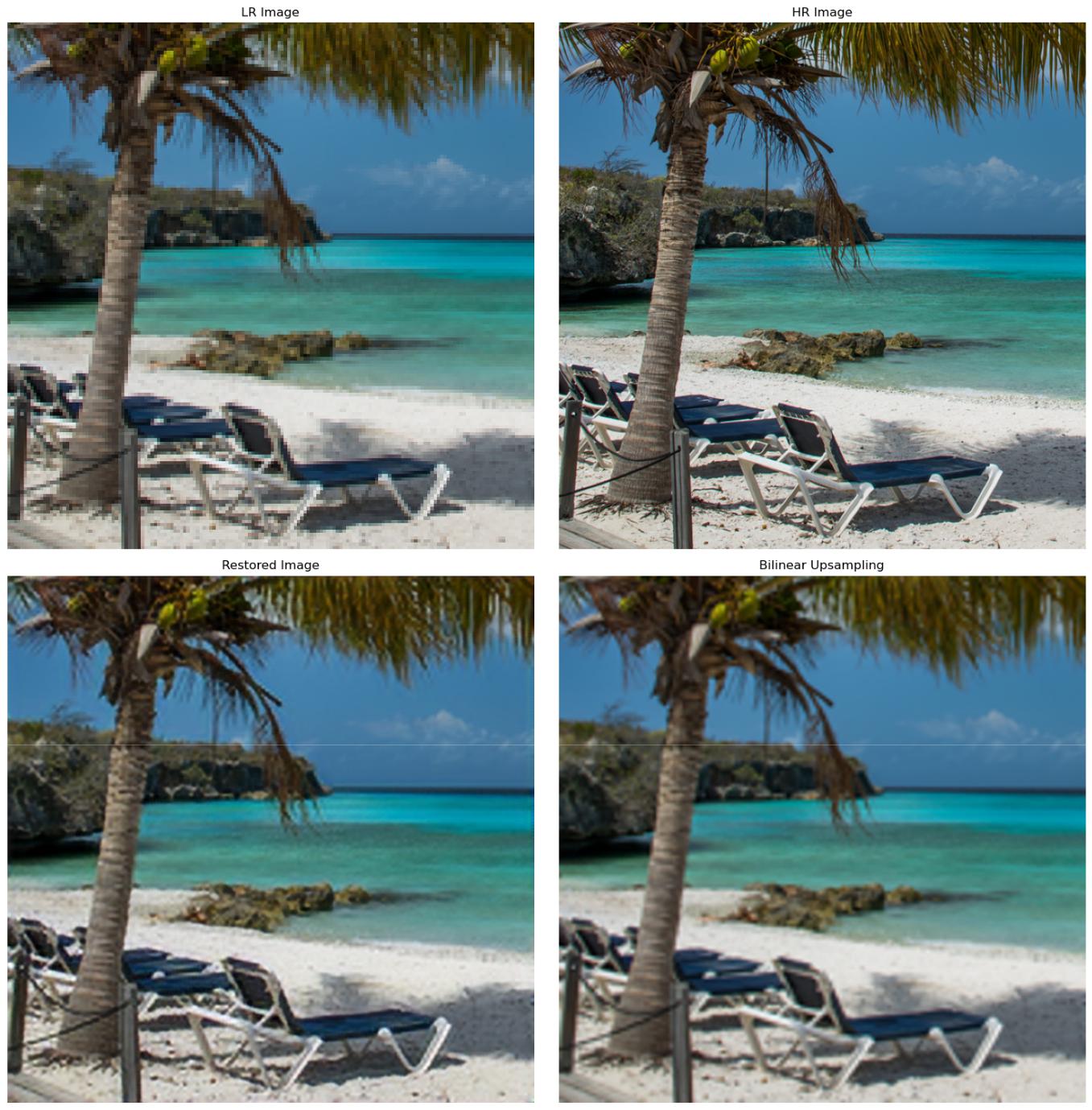
plt.title("Bilinear Upsampling")

​

plt.tight\_layout()

plt.show()

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



**Вывод**

При использовании модели FSRCNN (Fast Super-Resolution Convolutional Neural Network) для восстановления изображений возникает несколько ключевых выводов:

1. **Улучшение качества изображений**: FSRCNN предназначена для повышения разрешения изображений с минимальным уровнем искажений. Путем обучения на большом объеме данных она способна восстанавливать детали и текстуры в изображениях с низким разрешением, делая их более четкими и реалистичными.
2. **Быстродействие**: Как следует из названия, FSRCNN является быстрой моделью суперразрешения. Она представляет собой относительно легкую сеть, которая может быть эффективно выполнена на различных устройствах, включая мобильные устройства и встроенные системы.
3. **Подбор параметров**: Параметры модели, такие как число фильтров и слоев, могут быть подобраны с учетом требований к качеству и вычислительным ресурсам. Например, увеличение числа фильтров и слоев может улучшить качество восстановленных изображений, но может потребовать больше вычислительных ресурсов.
4. **Практическое применение**: FSRCNN может быть использована в различных областях, включая медицинское оборудование, видеосистемы наблюдения, обработку изображений и мультимедийные приложения. Ее способность повышать разрешение изображений может быть полезной в задачах, где важно получить более детальные изображения без увеличения шума или искажений.
5. **Оптимизация обучения**: При обучении модели FSRCNN важно правильно подобрать гиперпараметры, выбрать подходящую функцию потерь и использовать эффективные оптимизаторы для быстрого сходимости. Также может потребоваться использование регуляризации для предотвращения переобучения модели.

**Бибилиография**

1. Dong, C., Loy, C.C., He, K., Tang, X. (2016). Image Super-Resolution Using Deep Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2), 295-307. [Link (https://arxiv.org/abs/1608.00367)](https://arxiv.org/abs/1608.00367)
2. Saafke. (n.d.). FSRCNN\_Tensorflow. Retrieved from GitHub: [Link (https://github.com/Saafke/FSRCNN\_Tensorflow/tree/master?](https://github.com/Saafke/FSRCNN_Tensorflow/tree/master?tab=readme-ov-file) [tab=readme-ov-file)](https://github.com/Saafke/FSRCNN_Tensorflow/tree/master?tab=readme-ov-file)
3. Nhat-Thanh. (n.d.). SRCNN-TF. Retrieved from GitHub: [Link (https://github.com/Nhat-Thanh/SRCNN-TF/blob/main/README.md)](https://github.com/Nhat-Thanh/SRCNN-TF/blob/main/README.md)
4. Towards Data Science. (n.d.). A Review of FSRCNN for Super Resolution. Retrieved from Medium: [Link](https://towardsdatascience.com/review-fsrcnn-super-resolution-80ca2ee14da4) [(https://towardsdatascience.com/review-fsrcnn-super-resolution-80ca2ee14da4)](https://towardsdatascience.com/review-fsrcnn-super-resolution-80ca2ee14da4)
5. Википедия. (n.d.). Пиковое отношение сигнала к шуму. Retrieved from Wikipedia: [Link](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D0%BA%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BE%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB%D0%B0_%D0%BA_%D1%88%D1%83%D0%BC%D1%83) [(https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D0%BA%D0%BE%D0%B2%D0%BE%D0%B5\_%D0%BE%D1%82%D0%BD%D0%](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D0%BA%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BE%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB%D0%B0_%D0%BA_%D1%88%D1%83%D0%BC%D1%83)
6. Habr. (n.d.). Сверточные нейронные сети: свертка и пулинг. Retrieved from Habr: [Link (https://habr.com/ru/articles/111402/)](https://habr.com/ru/articles/111402/)



Ввод [ ]: ​

Ввод [ ]: ​



Ввод [ ]: ​

**По данной ссылке вы можете наблюдать весь результат работы** [**Link (https://drive.google.com/drive/folders/1u-0-6MObaxYEX6oBP6OU-tZJ9ne5BEZG?usp=drive\_link)**](https://drive.google.com/drive/folders/1u-0-6MObaxYEX6oBP6OU-tZJ9ne5BEZG?usp=drive_link)