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). 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.
2. **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.
3. **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.
4. **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.
5. **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/)

`

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) 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]:
```python
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(
    d: int,
    s: int,
    m: 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).
3. **`COLOR_CHANNELS`** : Количество каналов цветности в изображении. В RGB изображениях это обычно 3.
4. **`HR_IMG_SIZE`** : Размер изображений высокого разрешения (HR). Задается в виде кортежа `(width, height)` и выбирается на основе самого маленького изображения в наборе данных.
5. **`UPSCALING_FACTOR`** : Множитель увеличения, определяющий во сколько раз увеличивается размер изображений при применении модели. В данном случае, это 4.
6. **`LR_IMG_SIZE`** : Размер изображений низкого разрешения (LR), рассчитанный из размера изображений высокого разрешения и множителя увеличения.

Ввод [16]:
```python
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:

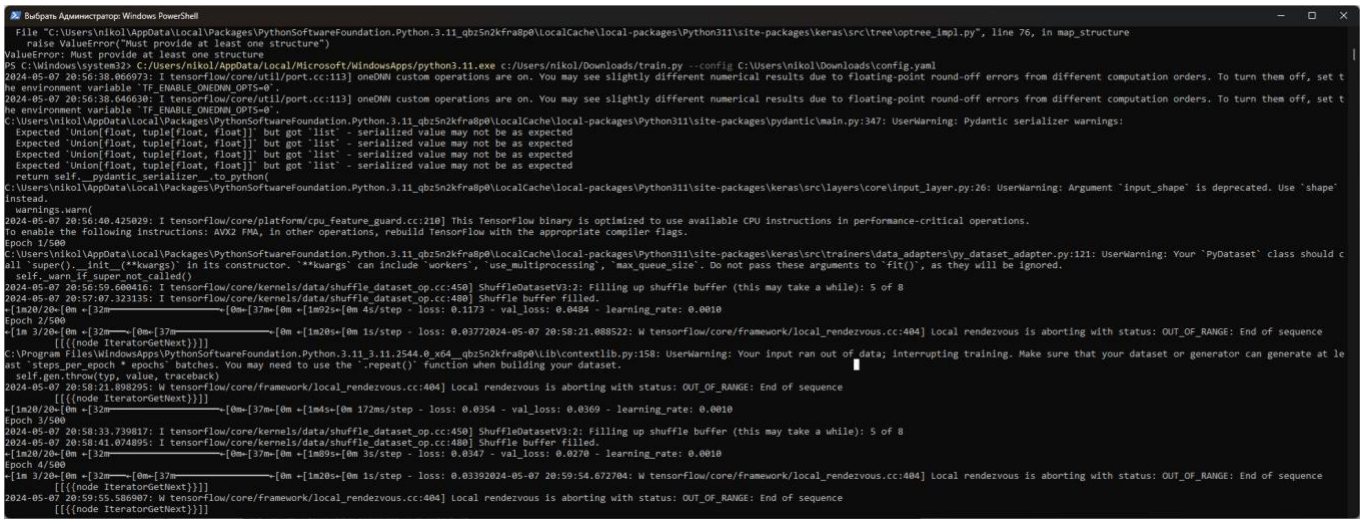
1. **Загрузка конфигурации**: Считывается конфигурационный файл YAML, который содержит параметры обучения модели, такие как пути к данным, параметры модели, оптимизаторы, количество эпох и так далее.
2. **Подготовка данных**: Создаются объекты наборов данных для обучения и валидации на основе класса `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 [ ]:
```yaml
# %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]:
```python
# 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).

$$PSNR = 10 \log_{10} \left(\frac{MAX_I^2}{MSE} \right) = 20 \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right)$$

Test Set

Ввод [6]:
```python
 # 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
```

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).
```

LR Image

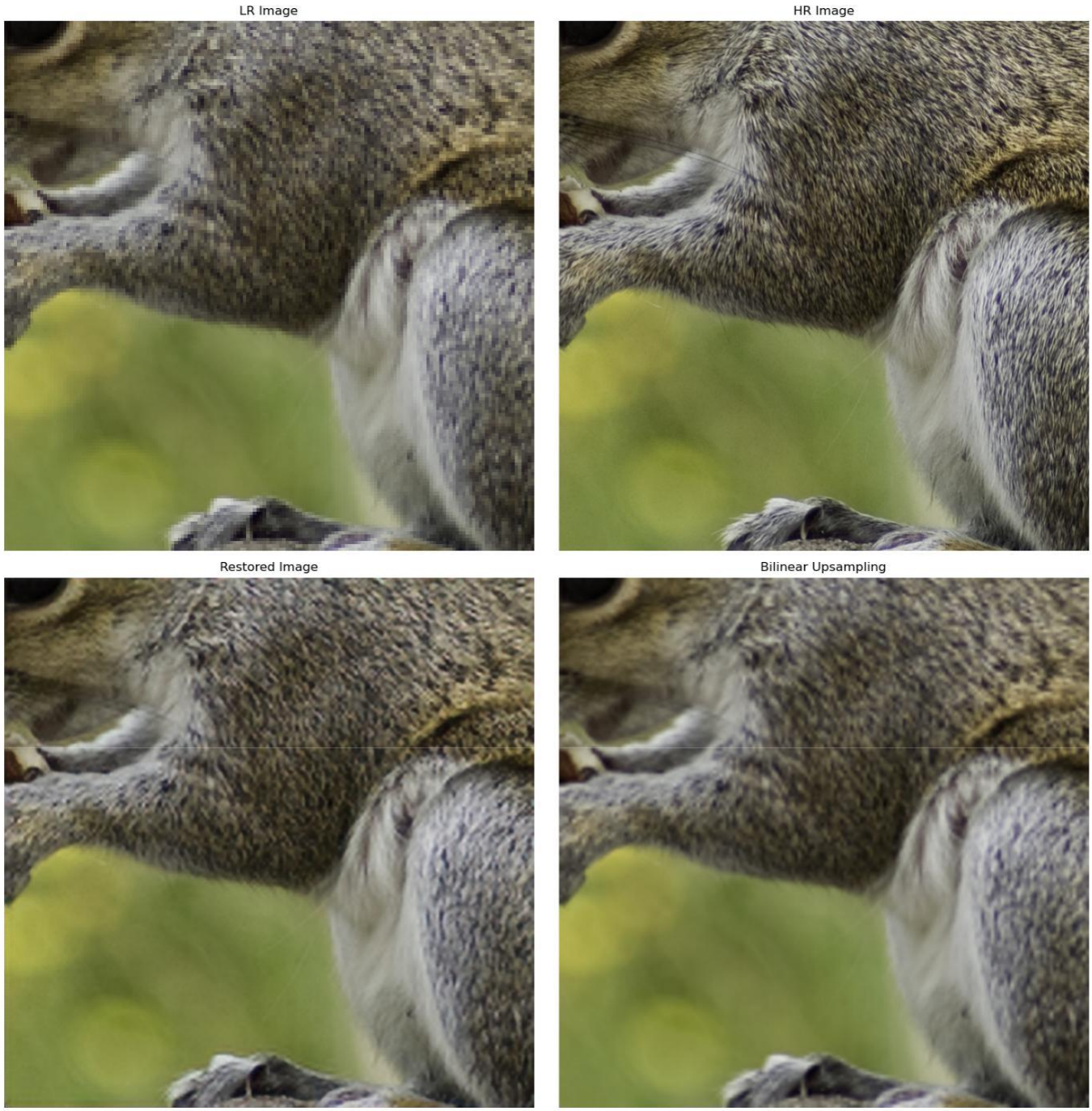
HR Image

Restored Image

Bilinear Upsampling

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).
```

LR Image

HR Image

Restored Image

Bilinear Upsampling

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).
```

LR Image

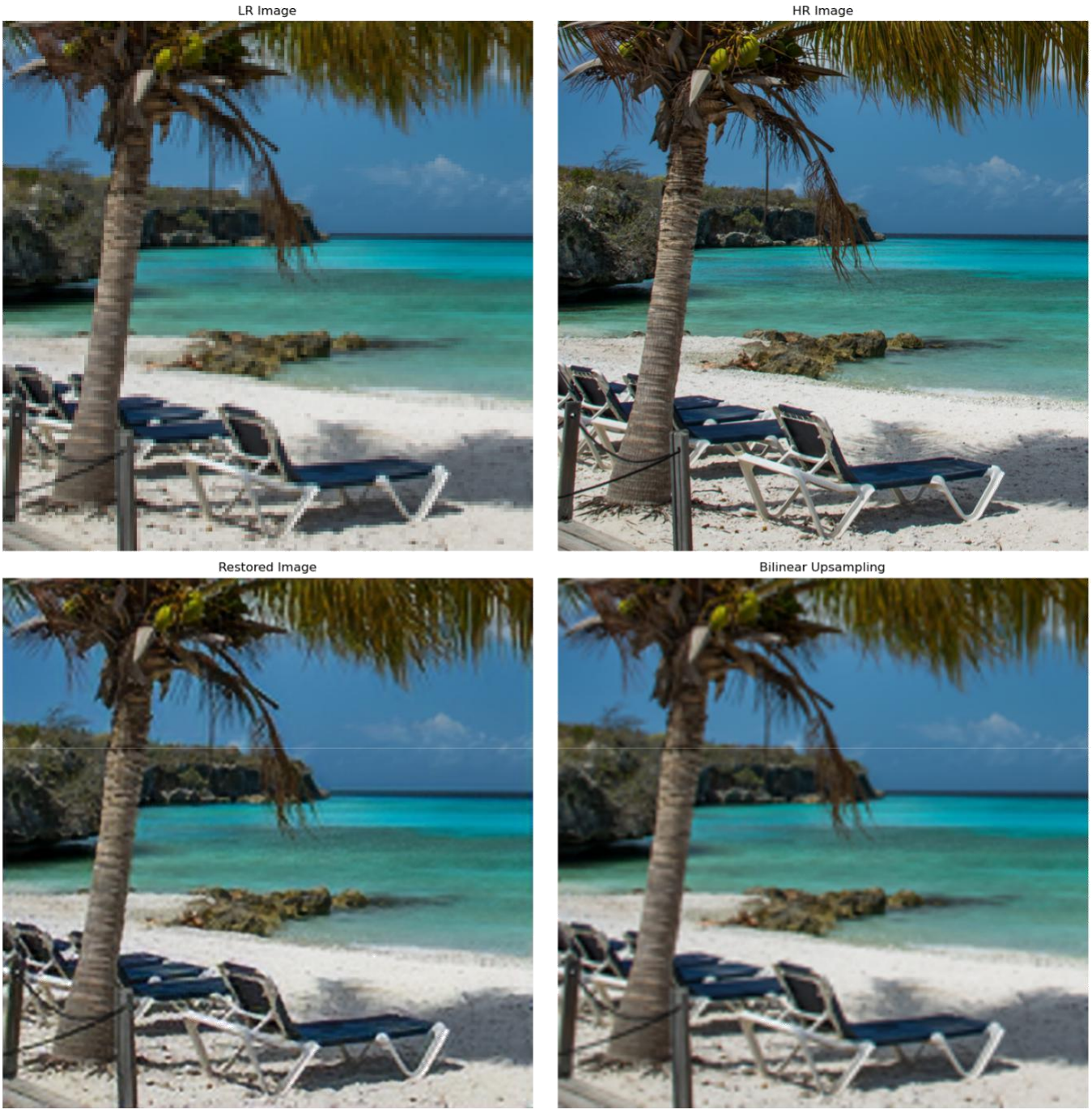
HR Image

Restored Image

Bilinear Upsampling

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)
2. Saafke. (n.d.). FSRCNN_Tensorflow. Retrieved from GitHub: Link (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)
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)
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%
6. Habr. (n.d.). Сверточные нейронные сети: свертка и пулинг. Retrieved from Habr: Link (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)