Курсовая работа

По дисциплине "Машинное обучение"

На тему: "Предварительный анализ данных и построение признаков в задачах повышения качества изображений"

Выполнил студент ПМ22-1: Хатуев Амаль

Руководитель: Одинцова Вера Александровна

Акутальность: В большинстве случаев файлообменники, социальные сети и другие платформы снижают качество изображений для экономии места на серверах. Это может привести к значительной потере качества, что создает потребность в восстановлении исходного вида. Ручная обработка графических художников становится дорогостоящей и неэффективной при обработке больших объемов данных. Использование моделей машинного обучения представляет собой более экономичный и масштабируемый подход, способный поддерживать высокое качество обработки при обработке больших объемов изображений.

```
import os
import re
from scipy import ndimage, misc
from tqdm import tqdm
from tensorflow.keras.preprocessing.image import img_to_array


from skimage.transform import resize, rescale
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(0)
import cv2 as cv2

import tensorflow as tf
from tensorflow.keras.layers import Input,
Dense ,Conv2D,MaxPooling2D ,Dropout
from tensorflow.keras.layers import Conv2DTranspose, UpSampling2D, add
from tensorflow.keras.models import Model
from keras import layers
from tensorflow.keras.utils import plot_model
from tensorflow.keras import regularizers
from tensorflow.keras.utils import plot_model
import tensorflow as tf
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())

print(tf.__version__)
```

```
2024-05-08 16:43:30.304598: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable
to register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
2024-05-08 16:43:30.304711: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
2024-05-08 16:43:30.430410: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable
to register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered

[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 13942927499207698474
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 16274030592
locality {
  bus_id: 1
  links {
  }
}
incarnation: 15637612784055736355
physical_device_desc: "device: 0, name: Tesla P100-PCIE-16GB, pci bus
id: 0000:00:04.0, compute capability: 6.0"
xla_global_id: 416903419
]
2.15.0
```

```
import tensorflow as tf
print(tf.__version__)

import keras
print(keras.__version__)
```

```
2.15.0
3.2.1
```

Подготовка датасета с изображениями

```
def sorted_alphanumeric(data):
    convert = lambda text: int(text) if text.isdigit() else
text.lower()
    alphanum_key = lambda key: [convert(c) for c in re.split('([0-
```

```
9]+)',key)]
    return sorted(data,key = alphanum_key)

SIZE = 256
high_img = []
path = '/kaggle/input/image-super-resolution/dataset/Raw
Data/high_res'
files = os.listdir(path)
files = sorted_alphanumeric(files)
for i in tqdm(files):
    if i == '855.jpg':
        break
    else:
        img = cv2.imread(path + '/'+i,1)
        # open cv reads images in BGR format so we have to convert it
to RGB
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        #resizing image
        img = cv2.resize(img, (SIZE, SIZE))
        img = img.astype('float32') / 255.0
        high_img.append(img_to_array(img))

low_img = []
path = '/kaggle/input/image-super-resolution/dataset/Raw Data/low_res'
files = os.listdir(path)
files = sorted_alphanumeric(files)
for i in tqdm(files):
    if i == '855.jpg':
        break
    else:
        img = cv2.imread(path + '/'+i,1)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        #resizing image
        img = cv2.resize(img, (SIZE, SIZE))
        img = img.astype('float32') / 255.0
        low_img.append(img_to_array(img))
```

```
100%|██████████| 855/855 [00:11<00:00, 75.17it/s]
100%|██████████| 855/855 [00:10<00:00, 78.34it/s]
```

```
for i in range(4):
    a = np.random.randint(0,855)
    plt.figure(figsize=(10,10))
    plt.subplot(1,2,1)
    plt.title('High Resolution Image', color = 'green', fontsize = 20)
    plt.imshow(high_img[a])
    plt.axis('off')
    plt.subplot(1,2,2)
    plt.title('low Resolution Image ', color = 'black', fontsize = 20)
```

```
    plt.imshow(low_img[a])
    plt.axis('off')
```

High Resolution Image

low Resolution Image

High Resolution Image

low Resolution Image

В данном датасете находятся два вида одних и тех же изображений: с высоким качеством и низким

Разделим выборку на train, test, val

```
train_high_image = high_img[:700]
train_low_image = low_img[:700]
train_high_image = np.reshape(train_high_image,
(len(train_high_image),SIZE,SIZE,3))
train_low_image = np.reshape(train_low_image,
(len(train_low_image),SIZE,SIZE,3))
```

```
val_high_image = high_img[700:830]
val_low_image = low_img[700:830]
val_high_image= np.reshape(val_high_image,
(len(val_high_image),SIZE,SIZE,3))
val_low_image = np.reshape(val_low_image,
(len(val_low_image),SIZE,SIZE,3))

test_high_image = high_img[830:]
test_low_image = low_img[830:]
test_high_image= np.reshape(test_high_image,
(len(test_high_image),SIZE,SIZE,3))
test_low_image = np.reshape(test_low_image,
(len(test_low_image),SIZE,SIZE,3))

print("Shape of training images:",train_high_image.shape)
print("Shape of test images:",test_high_image.shape)
print("Shape of validation images:",val_high_image.shape)
```

```
Shape of training images: (700, 256, 256, 3)
Shape of test images: (25, 256, 256, 3)
Shape of validation images: (130, 256, 256, 3)
```

Тренировочный набор: Содержит 700 изображений высокого разрешения и их соответствующие версии низкого разрешения. Каждое изображение имеет размеры 256x256 пикселей с 3 цветовыми каналами (RGB).

Валидационный набор: Состоит из 130 изображений высокого разрешения и их версий низкого разрешения. Эти изображения также имеют размеры 256x256 пикселей с 3 цветовыми каналами.

Тестовый набор: Включает 25 изображений высокого разрешения и их версии низкого разрешения. Опять же, изображения имеют размеры 256x256 пикселей с 3 цветовыми каналами.

Построение модели

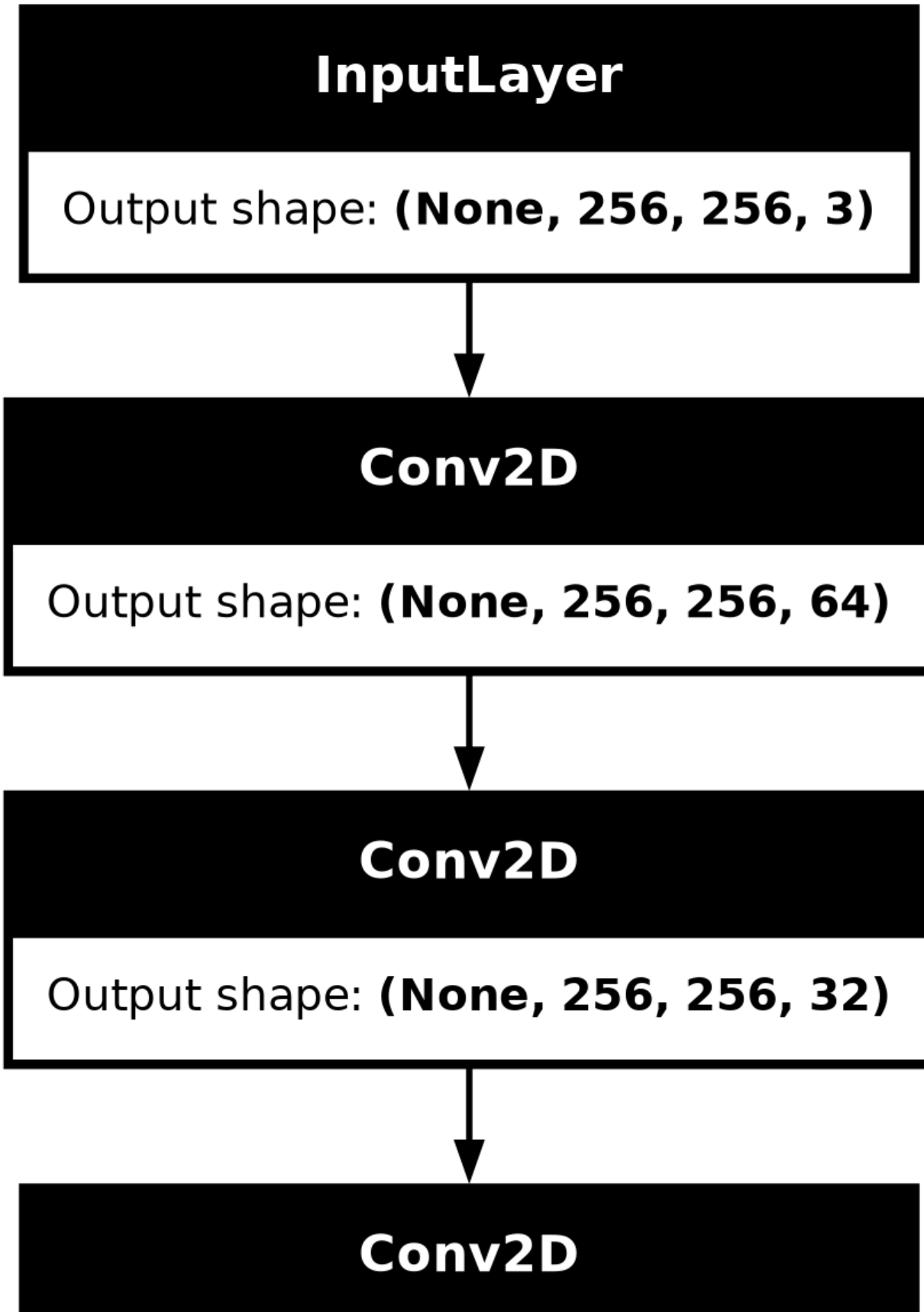
```
input_img = Input(shape=(256,256,3))
l1 = tf.keras.layers.Conv2D(64, 9, padding = 'same', activation =
'relu')(input_img)
l2 = tf.keras.layers.Conv2D(32, 1, padding = 'same', activation =
'relu')(l1)
l3 = tf.keras.layers.Conv2D(3, 5, padding = 'same', activation =
'relu')(l2)

SRCNN = Model(input_img, l3)

def pixel_mse_loss(x,y):
    return tf.reduce_mean( (x - y) ** 2 )
```

```
SRCNN.compile(optimizer=tf.keras.optimizers.Adam(0.001),loss=pixel_mse
_loss)
SRCNN.summary()
plot_model(SRCNN, to_file ='super_res.png',show_shapes=True)
```

```
Model: "functional_1"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ input_layer (InputLayer)             │ (None, 256, 256, 3)         │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d (Conv2D)                      │ (None, 256, 256, 64)        │
15,616 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d_1 (Conv2D)                    │ (None, 256, 256, 32)        │
2,080 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d_2 (Conv2D)                    │ (None, 256, 256, 3)         │
2,403 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────
┘
```

```
 Total params: 20,099 (78.51 KB)

 Trainable params: 20,099 (78.51 KB)

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


InputLayer
Output shape: (None, 256, 256, 3)
Conv2D
Output shape: (None, 256, 256, 64)
Conv2D
Output shape: (None, 256, 256, 32)
Conv2D

```
SRCNN.fit(train_low_image, train_high_image, epochs = 100, batch_size
= 1,
          validation_data = (val_low_image,val_high_image))

Epoch 1/100
 31/700 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 0.1142

WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1715186655.846902     115 device_compiler.h:186] Compiled
cluster using XLA!  This line is logged at most once for the lifetime
of the process.

700/700 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - loss: 0.0202 - val_loss:
0.0047
Epoch 2/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0030 - val_loss:
0.0019
Epoch 3/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.0027 - val_loss:
0.0017
Epoch 4/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0017 - val_loss:
0.0016
Epoch 5/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0037 - val_loss:
0.0032
Epoch 6/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0028 - val_loss:
0.0025
Epoch 7/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0016 - val_loss:
0.0014
Epoch 8/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0024 - val_loss:
0.0019
Epoch 9/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0016 - val_loss:
0.0016
Epoch 10/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.0020 - val_loss:
0.0015
Epoch 11/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.0014 - val_loss:
0.0013
Epoch 12/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.0015 - val_loss:
0.0015
Epoch 13/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0013 - val_loss:
```

```
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0010 - val_loss:
0.0010
Epoch 96/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0011 - val_loss:
0.0018
Epoch 97/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0012 - val_loss:
0.0011
Epoch 98/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0010 - val_loss:
0.0011
Epoch 99/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0011 - val_loss:
0.0010
Epoch 100/100
700/700 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0012 - val_loss:
0.0011

<keras.src.callbacks.history.History at 0x7f1388598c10>
```

Здесь я создаю и обучаю нейронную сеть SRCNN для выполнения задачи суперразрешения изображений.

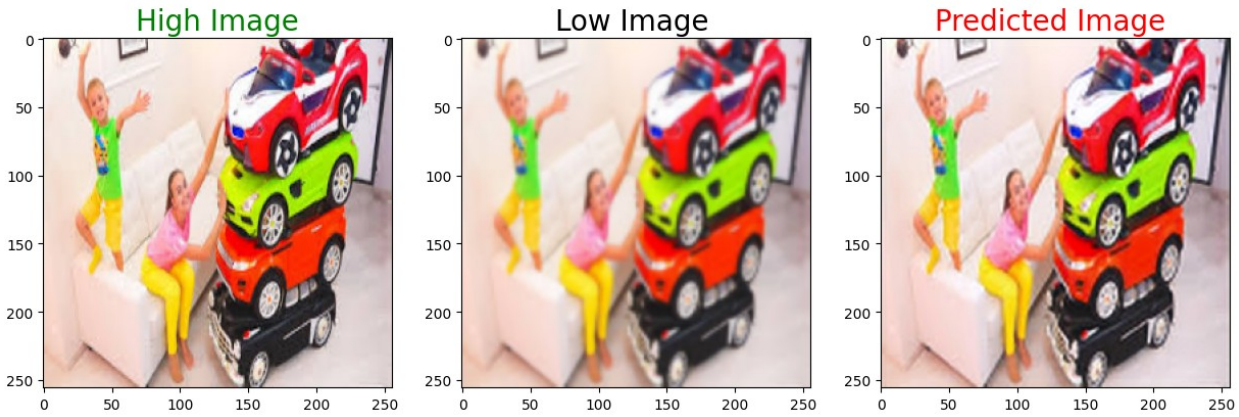
SRCNN использует сверточные слои для изучения пространственных зависимостей в изображениях и генерации суперразрешенных версий входных изображений.

Обучение модели производится на тренировочных данных, а ее производительность оценивается на валидационном наборе.

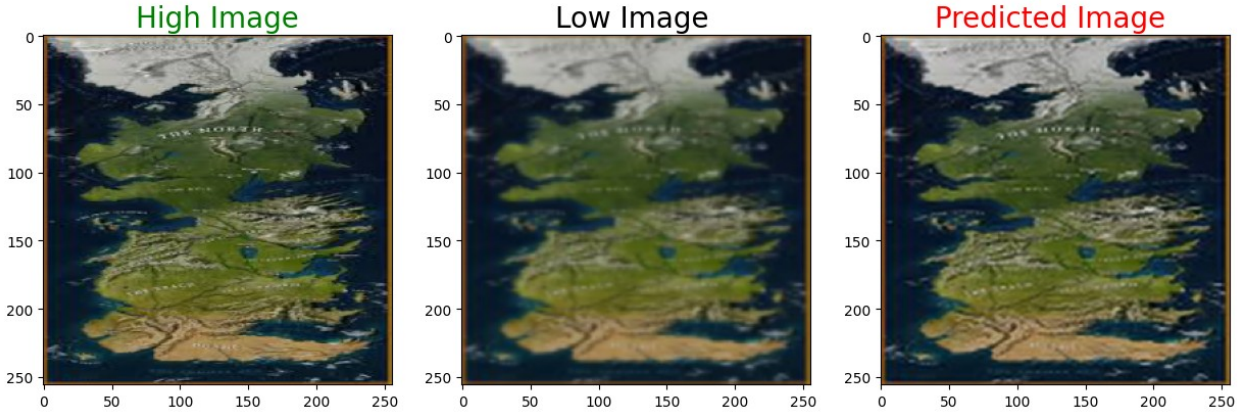
Функция потерь MSE используется для измерения различия между предсказанными и фактическими изображениями.

```
def PSNR(y_true,y_pred):
    mse=tf.reduce_mean( (y_true - y_pred) ** 2 )
    return 20 * log10(1/ (mse ** 0.5))

def log10(x):
    numerator = tf.math.log(x)
    denominator = tf.math.log(tf.constant(10, dtype=numerator.dtype))
    return numerator / denominator

def pixel_MSE(y_true,y_pred):
    return tf.reduce_mean( (y_true - y_pred) ** 2 )

def plot_images(high,low,predicted):
    plt.figure(figsize=(15,15))
    plt.subplot(1,3,1)
    plt.title('High Image', color = 'green', fontsize = 20)
    plt.imshow(high)
    plt.subplot(1,3,2)
```

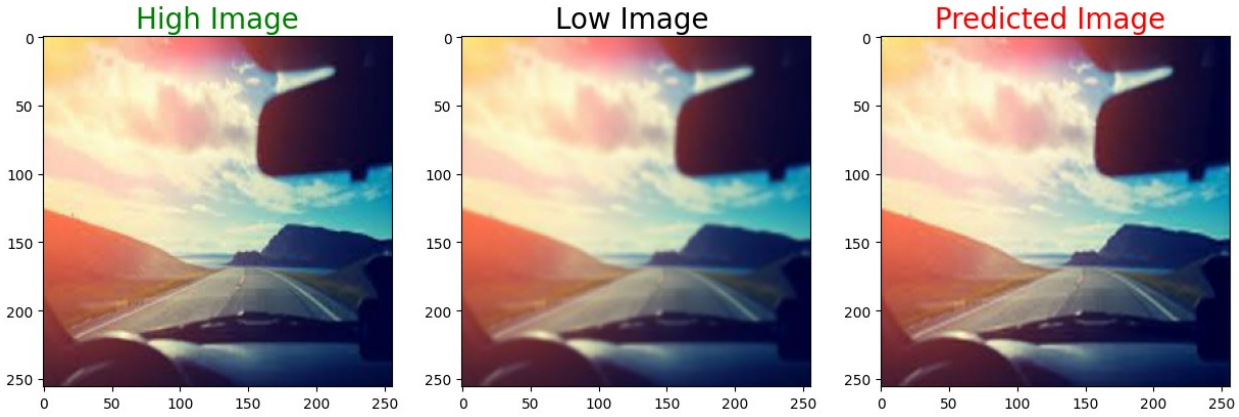
```
    plt.title('Low Image ', color = 'black', fontsize = 20)
    plt.imshow(low)
    plt.subplot(1,3,3)
    plt.title('Predicted Image ', color = 'Red', fontsize = 20)
    plt.imshow(predicted)

    plt.show()

for i in range(16,25):

    predicted =
np.clip(SRCNN.predict(test_low_image[i].reshape(1,SIZE,
SIZE,3)),0.0,1.0).reshape(SIZE, SIZE,3)
    plot_images(test_high_image[i],test_low_image[i],predicted)
    print('PSNR',PSNR(test_high_image[i],predicted),'dB',
"SSIM",tf.image.ssim(test_high_image[i],predicted,max_val=1))

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```




```
PSNR tf.Tensor(27.96211, shape=(), dtype=float32) dB SSIM
tf.Tensor(0.91987234, shape=(), dtype=float32)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

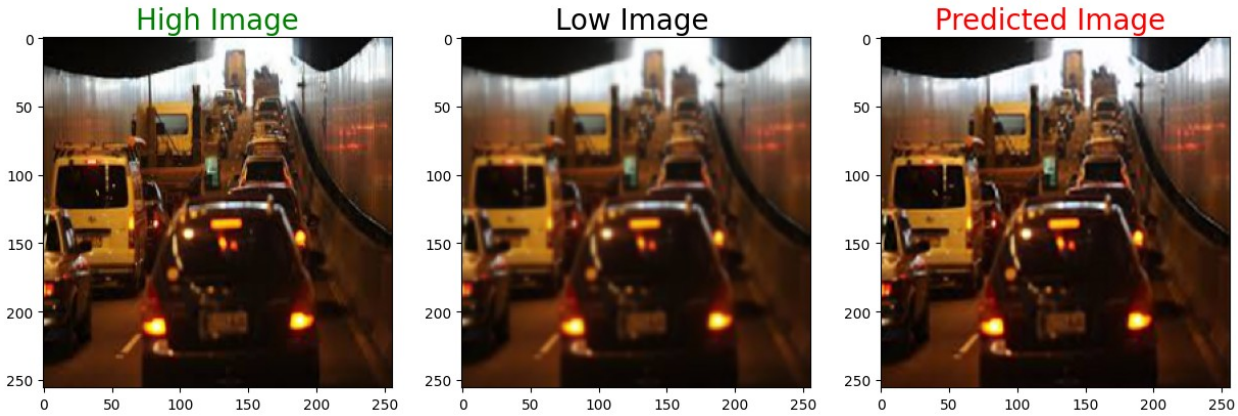


```
PSNR tf.Tensor(28.87555, shape=(), dtype=float32) dB SSIM
tf.Tensor(0.8711656, shape=(), dtype=float32)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

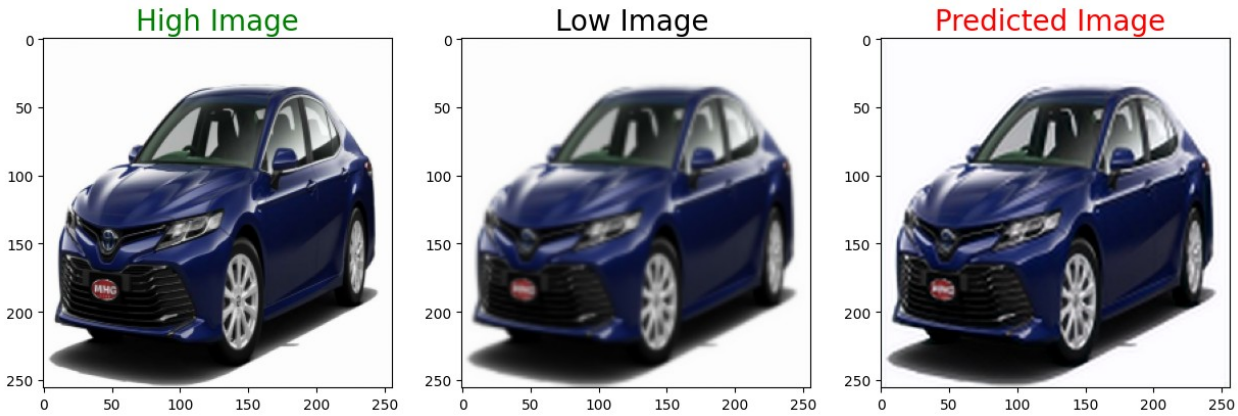



```
PSNR tf.Tensor(37.34618, shape=(), dtype=float32) dB SSIM
tf.Tensor(0.9710765, shape=(), dtype=float32)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```


```
PSNR tf.Tensor(29.152397, shape=(), dtype=float32) dB SSIM
tf.Tensor(0.8999775, shape=(), dtype=float32)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```



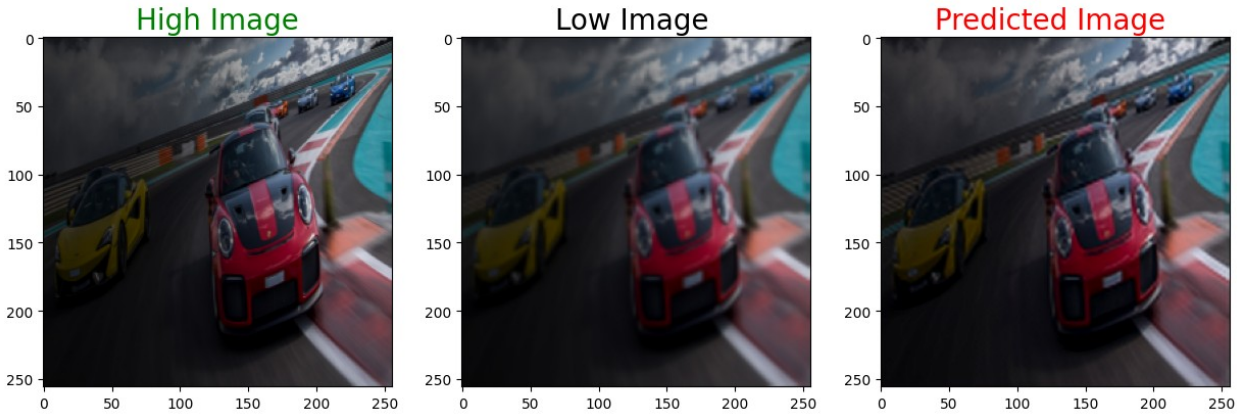

```
PSNR tf.Tensor(31.989994, shape=(), dtype=float32) dB SSIM
tf.Tensor(0.945203, shape=(), dtype=float32)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```




```
PSNR tf.Tensor(29.215916, shape=(), dtype=float32) dB SSIM
tf.Tensor(0.9406956, shape=(), dtype=float32)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

```
PSNR tf.Tensor(29.465443, shape=(), dtype=float32) dB SSIM
tf.Tensor(0.94587564, shape=(), dtype=float32)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```


```
PSNR tf.Tensor(27.612766, shape=(), dtype=float32) dB SSIM
tf.Tensor(0.91395617, shape=(), dtype=float32)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```




```
PSNR tf.Tensor(33.54394, shape=(), dtype=float32) dB SSIM
tf.Tensor(0.94290525, shape=(), dtype=float32)
```

Исходя из значений PSNR и SSIM, можно сделать следующие выводы о модели:

PSNR (Peak Signal-to-Noise Ratio): Значение PSNR составляет около 27 - 40 дБ. Это говорит о том, что среднее отношение сигнал-шум на предсказанных изображениях является довольно высоким. Чем выше это значение, тем меньше искажений содержится в предсказанных изображениях по сравнению с исходными.

SSIM (Structural Similarity Index): Значение SSIM около 0.92. Это указывает на высокое структурное сходство между предсказанными и исходными изображениями. Чем ближе SSIM к 1, тем более сходны изображения по структуре.

Таким образом, модель SRCNN демонстрирует хорошие результаты восстановления изображений на тестовом наборе данных низкого разрешения, выраженные как высокие значения PSNR и SSIM. Это говорит о том, что модель успешно улучшает качество изображений при суперразрешении.

```
# Save the entire model as a SavedModel.
# !mkdir -p saved_model
# SRCNN.save('/kaggle/working/')
# SRCNN.export('/kaggle/working/saved_model')
```

```
Saved artifact at '/kaggle/working/saved_model'. The following
endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 256, 256, 3),
dtype=tf.float32, name='keras_tensor')
Output Type:
  TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None)
Captures:
  139721864318992: TensorSpec(shape=(), dtype=tf.resource, name=None)
  139721870036128: TensorSpec(shape=(), dtype=tf.resource, name=None)
  139721864581840: TensorSpec(shape=(), dtype=tf.resource, name=None)
  139721864580784: TensorSpec(shape=(), dtype=tf.resource, name=None)
  139721864699696: TensorSpec(shape=(), dtype=tf.resource, name=None)
  139721864707088: TensorSpec(shape=(), dtype=tf.resource, name=None)
```