Федеральное государственное бюджетное образовательное учреждение высшего образования

«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»

Факультет информационных технологий и анализа больших данных

Департамент анализа данных и машинного обучения

КУРСОВАЯ РАБОТА

на тему:

«Предварительный анализ данных и построение признаков в задачах повышения качества изображений»

Выполнил(а):

студент(ка) группы ПМ22-4 факультета информационных технологий и анализа больших данных

Дьяков А.А.

Научный руководитель:

доцент, к.ф.-м.н. Одинцова В.А.

Москва 2024 г

Тема курсовой работы: Предварительный анализ данных и построение признаков в задачах повышения качества изображений

Цель данной курсовой работы - продемонстрировать основные методы анализа изображений, которые могут быть использованы для построения признаков в моделях улучшения качества изображений, а также показать преимущество специализированных моделей машинного обучения над простой интерполяцией при увеличении разрешения и улучшении качества изображений

Актуальность темы: В современном мире изображения играют важную роль в различных сферах, от медицины до развлечений. Часто изображения имеют низкое разрешение или качество, что затрудняет их использование. Методы машинного обучения позволяют значительно улучшить качество изображений, делая их более четкими, детализированными и информативными.

Задачи:

1. Провести анализ датасета:
   - Описать датасет, используемый в работе: https://www.kaggle.com/datasets/quadeer15sh/image-super-resolution-from-unsplash
   - Выделить особенности датасета: наличие изображений разного качества (оригинальных и ухудшенных), разнообразие сюжетов и объектов на фотографиях.
2. Рассмотреть и реализовать основные методы анализа изображений:
   - Гистограмма цветов: проанализировать распределение цветов в изображениях.
   - Градиенты: показать, как вычислять градиенты изображения и что они говорят о контурах и текстурах.
   - HOG (Histogram of Oriented Gradients): описать HOG и показать пример его применения.
   - Среднеквадратичное отклонение: показать, как использовать среднеквадратичное отклонение для оценки уровня шума в изображении.
   - Signal-to-Noise Ratio (SNR): объяснить, как SNR позволяет оценить соотношение между полезным сигналом и шумом в изображении.
3. Изучить и применить модели машинного обучения для улучшения качества изображений:
   - LapSRN_x8: описать модель, ее архитектуру и особенности. Показать пример ее применения для увеличения разрешения изображения.
   - EDSR_x4: описать модель, ее архитектуру и особенности. Показать пример ее применения для улучшения качества изображения.
4. Провести сравнительный анализ моделей с линейной интерполяцией:
   - Взять изображение низкого качества и увеличить его разрешение с помощью линейной интерполяции, LapSRN_x8 и EDSR_x4.
   - Оценить качество полученных изображений с помощью метрики BRISQUE.

- Сделать выводы о преимуществах моделей LapSRN_x8 и EDSR_x4 по сравнению с линейной интерполяцией.

5. Обобщить результаты проделанной работы.

Описание моделей:

LapSRN_x8 (Laplacian Pyramid Super-Resolution Network) и EDSR_x4 (Enhanced Deep Super-Resolution Network) - это две мощные нейросетевые модели, разработанные для увеличения разрешения и улучшения качества изображений.

1. LapSRN_x8:

LapSRN основана на лапласианской пирамиде, которая представляет изображение как набор разномасштабных деталей. Модель состоит из каскада CNN-модулей, каждый из которых восстанавливает детали на определенном уровне пирамиды. На каждом уровне пирамиды модель предсказывает остаточный образ, который добавляется к изображению более низкого разрешения для получения изображения более высокого разрешения.

Особенности:

- Прогрессивное восстановление: Модель постепенно восстанавливает детали изображения, начиная с низкого разрешения и переходя к высокому.
- Обучение с потерями на разных масштабах: Это позволяет модели более точно восстанавливать детали на всех уровнях пирамиды.
- Увеличение разрешения в x8: Как следует из названия, модель LapSRN_x8 предназначена для увеличения разрешения изображения в 8 раз.

2. EDSR_x4:

EDSR основана на глубокой CNN-архитектуре, которая состоит из множества остаточных блоков. Остаточные блоки позволяют модели изучать сложные зависимости между пикселями изображения. EDSR использует большие размеры ядра свертки, что позволяет модели захватывать больше контекстной информации.

Особенности:

- Упрощенная архитектура: EDSR отличается от других моделей super-resolution своей простой и эффективной архитектурой.
- Отсутствие пакетной нормализации: это упрощает обучение модели и снижает количество необходимых вычислительных ресурсов.
- Большие размеры ядер свертки: позволяют модели лучше учитывать контекст при восстановлении деталей.
- Увеличение разрешения в x4: EDSR_x4 предназначена для увеличения разрешения изображения в 4 раза.

In [26]:
```
import os
import shutil
from PIL import Image
import cv2
from cv2 import dnn_superres
import matplotlib.pyplot as plt
from skimage.feature import local_binary_pattern
from brisque import BRISQUE
import numpy as np
#from sewar import full_ref
```

In [2]:
```
#!pip3 install opencv-contrib-python==4.5.5.62
```

Датасет

https://www.kaggle.com/datasets/quadeer15sh/image-super-resolution-from-unsplash

в датасете множество фотографий. Фотографии с припиской *_2 - оригинальные фото, с припиской* _6 - те же изображения, но плохого качества

In [13]:
```
#предобработка датасета

# Путь к исходной папке
source_folder = r'C:\Users\Андрей\курсовая\low res'

# Путь к папке, которую вы хотите создать для сохранения фотографий
destination_folder = r'C:\Users\Андрей\курсовая\low res filtered'

if not os.path.exists(destination_folder):
    os.makedirs(destination_folder)

# Перебираем все файлы в исходной папке
for filename in os.listdir(source_folder):
    if filename.endswith("_6.jpg"):
        # Если файл заканчивается на "_6.jpg", копируем его в новую папку
        shutil.copy(os.path.join(source_folder, filename), destination_folder)

# Перебираем все файлы в исходной папке
for filename in os.listdir(source_folder):
    if filename.endswith("_2.jpg"):
        # Если файл заканчивается на "_2.jpg", копируем его в новую папку
        shutil.copy(os.path.join(source_folder, filename), destination_folder)
```

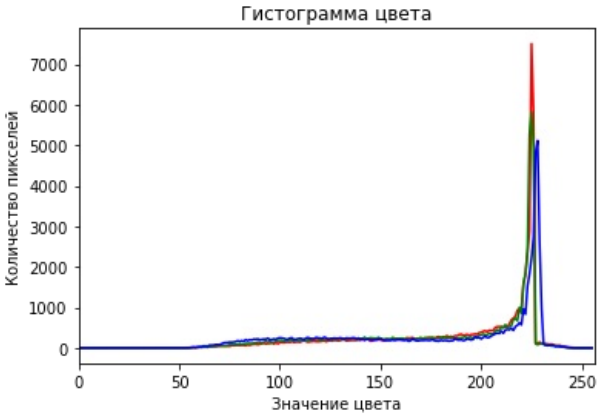
In [58]:
```
#уменьшение размера изображения
def resize(im, new_width):
    width, height = im.size
    ratio = height/width
    new_height = int(ratio*new_width)
    resized_image = im.resize((new_width,new_height))
    return resized_image

files = os.listdir("low res filtered")
extensions = ['jpg','png','jpeg']
for file in files:
    ext = file.split('.')[-1]
    if ext in extensions:
        im = Image.open('low res filtered/'+file)
        im_resized = resize(im, 300)
        new_dir = 'resized_images'
        if not os.path.exists(new_dir):
            os.makedirs(new_dir)
        file = f'{new_dir}/{file}'
        im_resized.save(file, 'png', quality=100)
```

Гистограмма цвета показывает распределение цветов в изображении. Она подсчитывает, сколько пикселей приходится на каждый цветовой диапазон.

In [50]:
```
def hist_colors(path):
    import cv2
    import numpy as np
    import matplotlib.pyplot as plt

    # Загрузка изображения
    image = cv2.imread(path)

    # Конвертация в RGB (OpenCV по умолчанию загружает в BGR)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    max_colors = list()

    # Вычисление гистограммы
    colors = ('r', 'g', 'b')
    for i, color in enumerate(colors):
        hist = cv2.calcHist([image], [i], None, [256], [0, 256])
        plt.plot(hist, color=color)
```

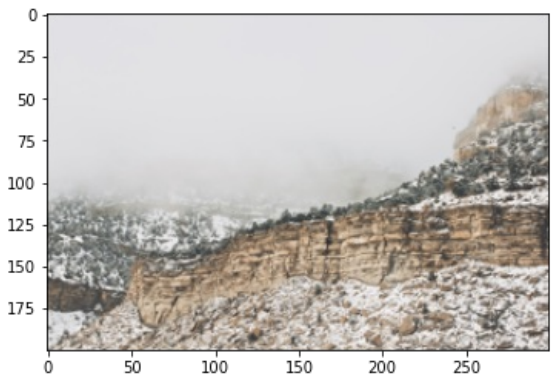
```
        plt.xlim([0, 256])
        max_colors.append(np.argmax(hist))

    plt.title('Гистограмма цвета')
    plt.xlabel('Значение цвета')
    plt.ylabel('Количество пикселей')
    plt.show()
    return max_colors
```

In [73]:
```
hist_colors('resized_images\\2_2.jpg')
```

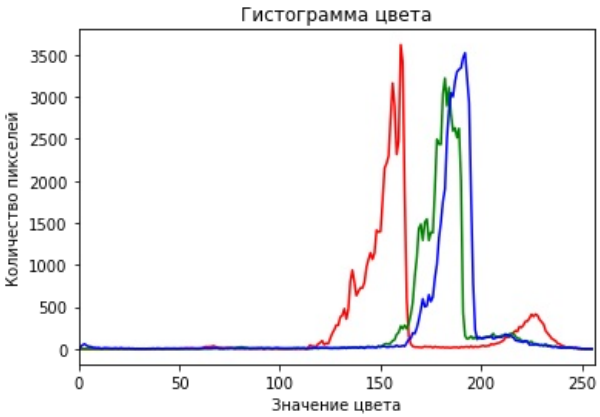


Out[73]:
```
[225, 225, 228]
```

In [74]:
```
plt.imshow(cv2.imread('resized_images\\2_2.jpg')[:,:,::-1]);
```



In [75]:
```
hist_colors('resized_images\\22_2.jpg')
```



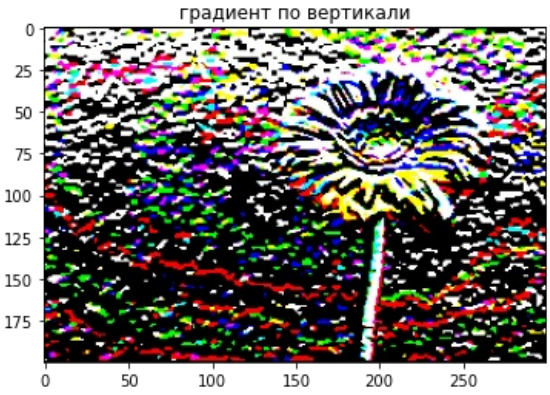
Out[75]:
```
[160, 182, 192]
```

In [76]:
```
plt.imshow(cv2.imread('resized_images\\22_2.jpg')[:,:,::-1]);
```

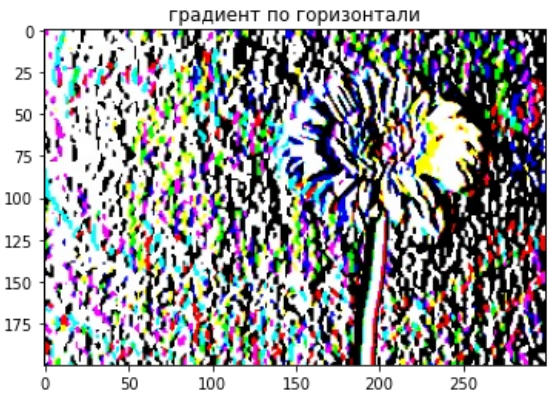
Градиенты показывают направление и величину изменения интенсивности пикселей. Они выделяют границы объектов, контуры и текстурные детали.

In [122…
```
# Вычисление градиентов
img = cv2.imread('resized_images\\22_2.jpg')
sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5) #градиент по горизонтали
sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5) #градиент по вертикали
plt.imshow(sobely)
plt.title("градиент по вертикали")
plt.show()
plt.imshow(sobelx)
plt.title("градиент по горизонтали")
plt.show()
```

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




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



HOG - Histogram of Oriented Gradients.
HOG представляет изображение как набор локальных гистограмм градиентов. Изображение делится на ячейки, в каждой ячейке строится гистограмма направлений градиентов.

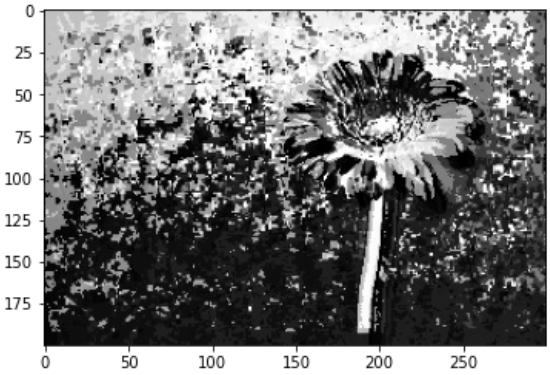
In [101…
```
# Вычисление HOG
hog = cv2.HOGDescriptor()
h = hog.compute(img)
h
```

Out[101]:
```
array([0.17969872, 0.08587766, 0.16998473, ..., 0.12023286, 0.04017976,
       0.13559937], dtype=float32)
```

LBP - Local Binary Patterns
LBP описывает текстуру, сравнивая каждый пиксель с его соседями. Если соседний пиксель ярче центрального, то ставим 1, иначе 0. Получаем бинарный код для каждого пикселя.

In [99]:
```
img = cv2.imread('resized_images\\22_2.jpg', cv2.IMREAD_GRAYSCALE)
```

```
# Вычисление LBP
# Параметры:
# P: количество точек выборки
# R: радиус круга
lbp = local_binary_pattern(img, P=8, R=10, method='default')

plt.imshow(lbp, cmap='gray')
plt.show()
```



Среднеквадратичное отклонение (Standard Deviation): Это стандартная метрика для измерения разброса значений яркости пикселей. Большое значение стандартного отклонения обычно указывает на большой уровень шума.

In [28]:
```
np.std(img)
```

Out[28]:
```
35.09114007034256
```

Signal-to-Noise Ratio (SNR): Отношение сигнал/шум. Это отношение между сигналом (то, что вы хотите увидеть) и шумом (то, что мешает его увидеть). Чем выше SNR, тем лучше.

In [29]:
```
mean_signal = np.mean(img)
std_noise = np.std(img - mean_signal)
snr = mean_signal / std_noise
snr
```

Out[29]:
```
1.2886894979731587
```

Модели

в датасете есть множество картинок, и у каждой картинки есть версия плохого и хорошего качества
Я беру картинку плохого качества, с помощью двух моделей улучшаю картинку (как в качестве, так и в расширении), получаю две картинки (result_edsr и result_lapsrn). Также увеличиваю расширение картинки с плохим качеством с помощью линейной интерполяции)
в итоге получаю 4 картиинки: исходную с хоошим качеством, улучшенную моделью LapSRN_x8, улучшенную моделью EDSR_x4, увеличенную линейной интерполяцией картинку с плохим качеством

In [32]:
```
path_img = "resized_images\\1_6.jpg"
#path_img = "low res\\1_6.jpg"
img = cv2.imread(path_img)
plt.imshow(img[:,:,::-1])
plt.show()
```

In [33]:
```
sr_lapsrn = cv2.dnn_superres.DnnSuperResImpl_create()

path_lapsrn = "LapSRN_x8.pb"

sr_lapsrn.readModel(path_lapsrn)

sr_lapsrn.setModel("lapsrn",8)
```

In [34]:
```
sr_edsr = cv2.dnn_superres.DnnSuperResImpl_create()

path_edsr = "EDSR_x4.pb"

sr_edsr.readModel(path_edsr)

sr_edsr.setModel("edsr",4)
```

In [35]:
```
result_edsr = sr_edsr.upsample(img) #размер был 200x300, стал 800x1200
```

In [36]:
```
result_lapsrn = sr_lapsrn.upsample(img)
```

In [25]:
```
cv2.imwrite("1_LapSRN.jpg", result_lapsrn) #сохранение
cv2.imwrite("1_edsr.jpg", result_edsr)
```

Out[25]:
```
True
```

In [73]:
```
original_im = cv2.imread("low res filtered\\1_2.jpg")
resized_bad_im = cv2.resize(img,dsize=None,fx=4,fy=4, interpolation=cv2.INTER_LINEAR) #было 200x300, стало 800x
```

In [74]:
```
#edsr увеличивает расширение в 4 раза по обеим осям, lapsrn в 8 раз соответсвенно
result_edsr.shape, result_lapsrn.shape, original_im.shape, resized_bad_im.shape
```

Out[74]:
```
((800, 1200, 3), (1600, 2400, 3), (800, 1200, 3), (800, 1200, 3))
```

In [75]:
```
fig, axs = plt.subplots(2, 2, figsize=(15, 10))

axs[0, 0].imshow(original_im[:,:,::-1])
axs[0, 0].set_title('Исходное изображение')

axs[0, 1].imshow(img[:,:,::-1])
axs[0, 1].set_title('Изображение с плохим качеством')

axs[1, 0].imshow(result_lapsrn[:,:,::-1])
axs[1, 0].set_title('Улучшенное lapsrn')

axs[1, 1].imshow(result_edsr[:,:,::-1])
axs[1, 1].set_title('Улучшенное edsr')

for ax in axs.flat:
    ax.axis('off')

plt.show()
```

Метрика качества изображения

BRISQUE score (Blind/Referenceless Image Spatial QUality Evaluator) - это показатель, используемый для оценки качества

изображения без необходимости сравнения с идеальным изображением. BRISQUE основан на изучении "естественных" искажений, которые присутствуют в изображениях, сделанных с помощью обычных камер.

Что означает BRISQUE score?
Низкий BRISQUE score (ближе к 0) указывает на высокое качество изображения. Изображение с низким BRISQUE score будет выглядеть четким, с хорошим контрастом и без заметных артефактов.
Высокий BRISQUE score указывает на низкое качество изображения. Изображение с высоким BRISQUE score может быть размытым, шумным или иметь другие искажения, которые делают его менее приятным для просмотра.

Преимущества BRISQUE score:

- Не требует эталонного изображения: BRISQUE может оценивать качество изображения без сравнения с идеальным изображением.
- Быстрый и эффективный: BRISQUE score можно рассчитать быстро и эффективно, что делает его полезным инструментом для оценки качества большого количества изображений.
- Хорошо коррелирует с субъективным восприятием: Исследования показывают, что BRISQUE score хорошо коррелирует с тем, как люди оценивают качество изображения.

In [4]:
```
#pip install brisque
```

```
Collecting brisque
  Downloading brisque-0.0.16-py3-none-any.whl (140 kB)
Requirement already satisfied: scikit-image in c:\anaconda\lib\site-packages (from brisque) (0.22.0)
Collecting libsvm-official
  Downloading libsvm_official-3.32.0-cp39-cp39-win_amd64.whl (52 kB)
Requirement already satisfied: scipy in c:\anaconda\lib\site-packages (from brisque) (1.10.1)
Requirement already satisfied: numpy in c:\anaconda\lib\site-packages (from brisque) (1.24.3)
Collecting opencv-python
  Using cached opencv_python-4.9.0.80-cp37-abi3-win_amd64.whl (38.6 MB)
Requirement already satisfied: pillow>=9.0.1 in c:\anaconda\lib\site-packages (from scikit-image->brisque) (9.0
.1)
Requirement already satisfied: lazy_loader>=0.3 in c:\anaconda\lib\site-packages (from scikit-image->brisque) (
0.4)
Requirement already satisfied: imageio>=2.27 in c:\anaconda\lib\site-packages (from scikit-image->brisque) (2.3
4.1)
Requirement already satisfied: packaging>=21 in c:\anaconda\lib\site-packages (from scikit-image->brisque) (21.
3)
Requirement already satisfied: tifffile>=2022.8.12 in c:\anaconda\lib\site-packages (from scikit-image->brisque
) (2024.5.10)
Requirement already satisfied: networkx>=2.8 in c:\anaconda\lib\site-packages (from scikit-image->brisque) (3.2
.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\anaconda\lib\site-packages (from packaging>=21->s
cikit-image->brisque) (3.0.4)
Installing collected packages: opencv-python, libsvm-official, brisque
Successfully installed brisque-0.0.16 libsvm-official-3.32.0 opencv-python-4.9.0.80
Note: you may need to restart the kernel to use updated packages.
```

In [19]:
```
BRISQUEscore = BRISQUE(url=False)
```

In [81]:
```
original_im.shape, resized_bad_im.shape, result_edsr.shape, result_lapsrn.shape
```

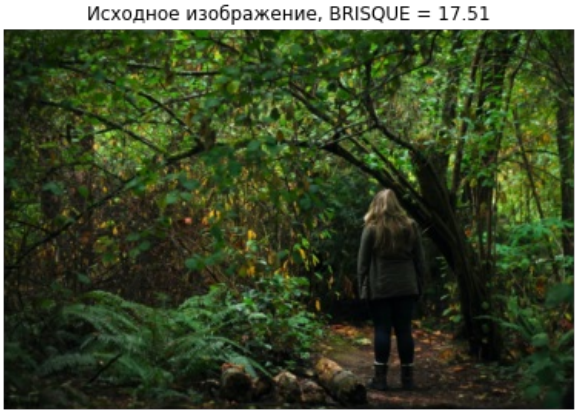
Out[81]:
```
((800, 1200, 3), (800, 1200, 3), (800, 1200, 3), (1600, 2400, 3))
```

In [79]:
```
BRISQUEscore.score(original_im), BRISQUEscore.score(resized_bad_im), BRISQUEscore.score(result_edsr), BRISQUEsc
```

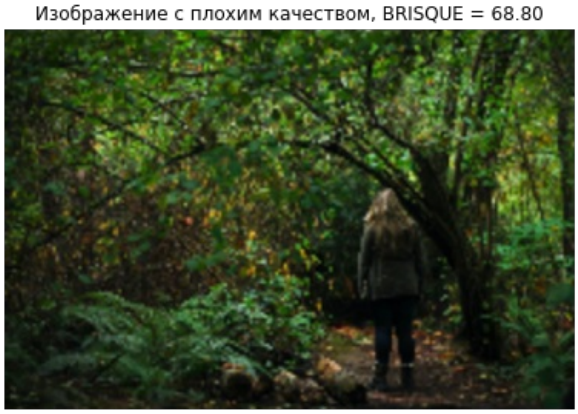
Out[79]:
```
(17.513771796179498, 68.79957558047656, 64.39132866840711, 55.981809330794675)
```

In [80]:
```
Bs_original = BRISQUEscore.score(original_im)
Bs_img = BRISQUEscore.score(resized_bad_im)
Bs_lapsrn = BRISQUEscore.score(result_lapsrn)
Bs_edsr = BRISQUEscore.score(result_edsr)
fig, axs = plt.subplots(2, 2, figsize=(15, 10))  # 2 строки, 2 столбца

axs[0, 0].imshow(original_im[:,:,::-1])
axs[0, 0].set_title(f'Исходное изображение, BRISQUE = {Bs_original:.2f}')

axs[0, 1].imshow(resized_bad_im[:,:,::-1])
axs[0, 1].set_title(f'Изображение с плохим качеством, BRISQUE = {Bs_img:.2f}')

axs[1, 0].imshow(result_lapsrn[:,:,::-1])
axs[1, 0].set_title(f'Улучшенное lapsrn, BRISQUE = {Bs_lapsrn:.2f}')

axs[1, 1].imshow(result_edsr[:,:,::-1])
axs[1, 1].set_title(f'Улучшенное edsr, BRISQUE = {Bs_edsr:.2f}')

for ax in axs.flat:
    ax.axis('off')

plt.show()
```

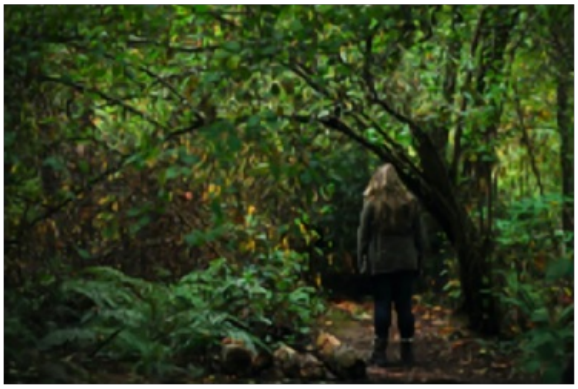
Исходное изображение, BRISQUE = 17.51



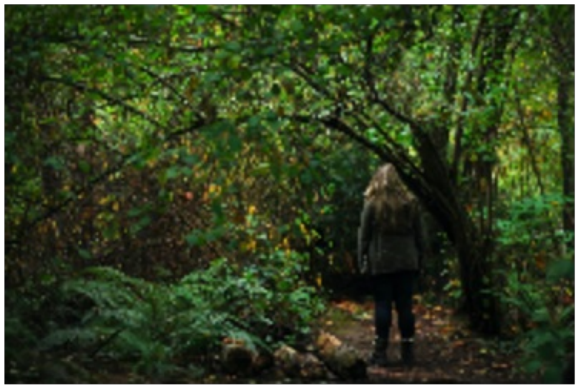
Изображение с плохим качеством, BRISQUE = 68.80



Улучшенное lapsrn, BRISQUE = 55.98

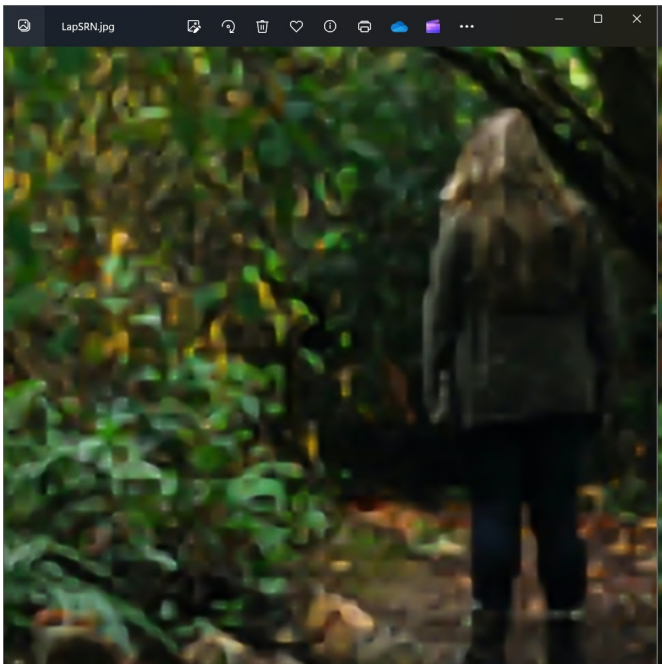


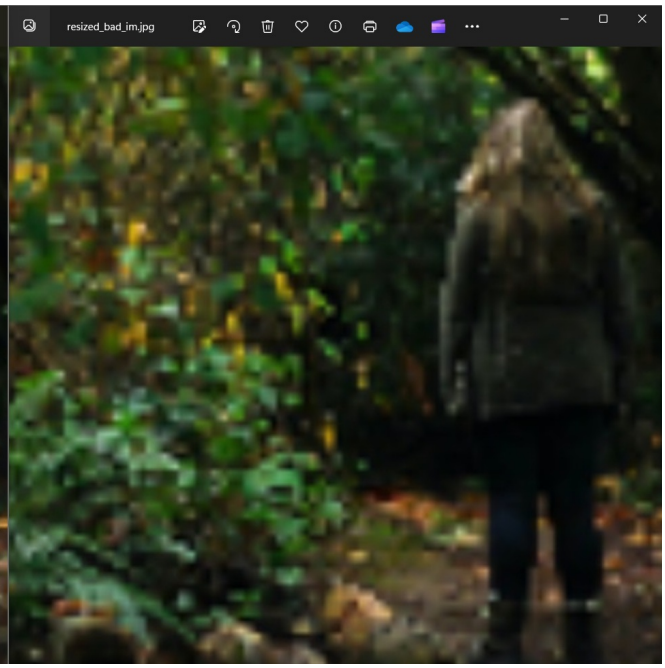
Улучшенное edsr, BRISQUE = 64.39

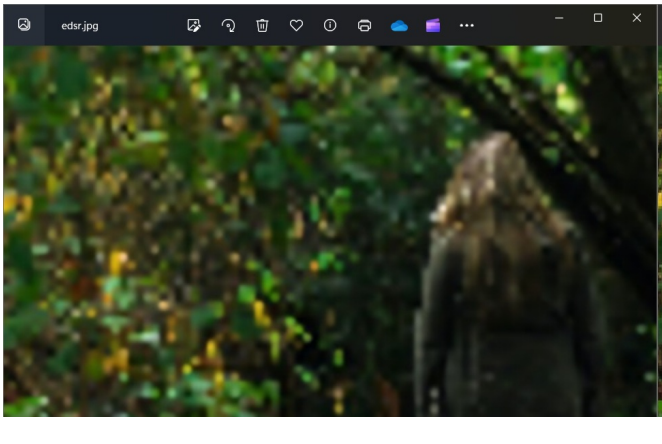


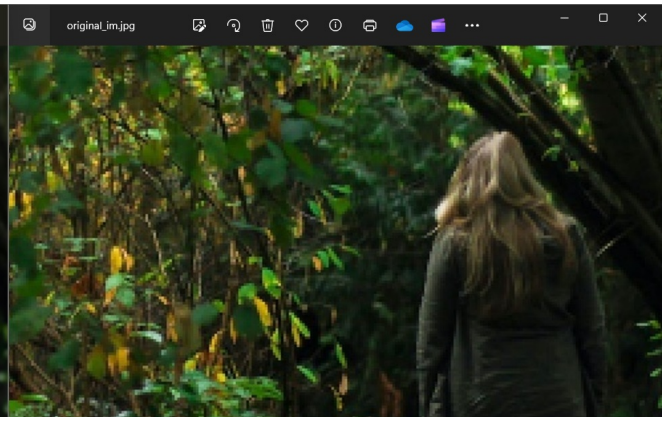
In [82]:
```
cv2.imwrite("original_im.jpg", original_im) #сохранение
cv2.imwrite("resized_bad_im.jpg", resized_bad_im)
cv2.imwrite("LapSRN.jpg", result_lapsrn)
cv2.imwrite("edsr.jpg", result_edsr)
```

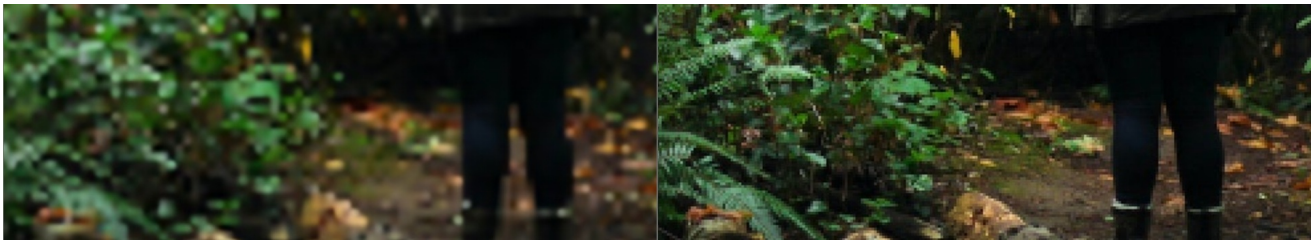
Out[82]:
```
True
```











Слева сверху картинка улучшена моделью LapSRN
Слева снизу картинка улучшена моделью EDSR
Справа сверху картинка плохого качества, увеличенная с помощью линейной интерполяции
Справа снизу исходная картинка хорошего качества