СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Устранение размытия изображения — это задача, в которой главной целью является устранение элементов, вызывающих размытость, и улучшение качества изображения для более ясной визуализации текстур и объектов [1], с другой точки зрения в задачи рассмотривается улучшение текстуру и качество изображений для дальнейшего использования в задачах машинного зрения, таких как обнаружение объектов и сегментация изображений, шум и атмосферные помехи, движение объектов, дрожание камеры и оборудование для расфокусировки являются распространенными источниками ухудшения качества изображения [2].

Устранение размытия изображения остается значимой задачей в сфере обработки изображений и компьютерного зрения. С развитием фотографии и видеосъемки возникает постоянная потребность в повышении качества изображений, особенно в условиях недостаточного освещения, движущихся объектов и других факторов, способных вызвать размытие. Методы улучшения четкости изображений постоянно совершенствуются, включая разработки в области машинного и глубокого обучения, что позволяет достигать более точных и эффективных результатов [3], Поэтому задача привлекает внимание исследователей и разработчиков программного обеспечения. Применение улучшенных изображений находит свое применение в различных областях, включая медицину, робототехнику, автомобильную промышленность и другие. Также стоит отметить, что улучшение качества изображений играет важную роль в раскрытии и расследовании преступлений. Более четкое изображение может существенно улучшить возможности идентификации объектов и лиц, что помогает в расследовании криминальных деяний [4].

Существует ряд алгоритмов, решающих данную задачу, но в последние десятилетия все разработанные методы, обычно используемые в области обработки изображений и компьютерного зрения, применяют в своих архитектурах нейронные сети и показывают наилучшие результаты по сравнению с классическими алгоритмами [5].

Цель данной преддипломной практики является разработка программно-алгоритмического комплекса метода устранения размытия изображения с применением нейронных сетей.

Задачи:

– Разработка программного обеспечения, метода и выбора корпусов данных;

– Обучение и валидация сети;

– Тестирование сети;

– Проведение анализа метода по объективным метрикам;

– Составление таблиц с результатами.

1 Технологический раздел

1.1. Формализация задачи

В данном подразделе рассмотрим формализацию задачи восстановления размытия изображений с применением нейронной сети.

На рисунке (1.1) представлено, формализация задачи восстановление размытие изображение нулевый уровень преобразований:

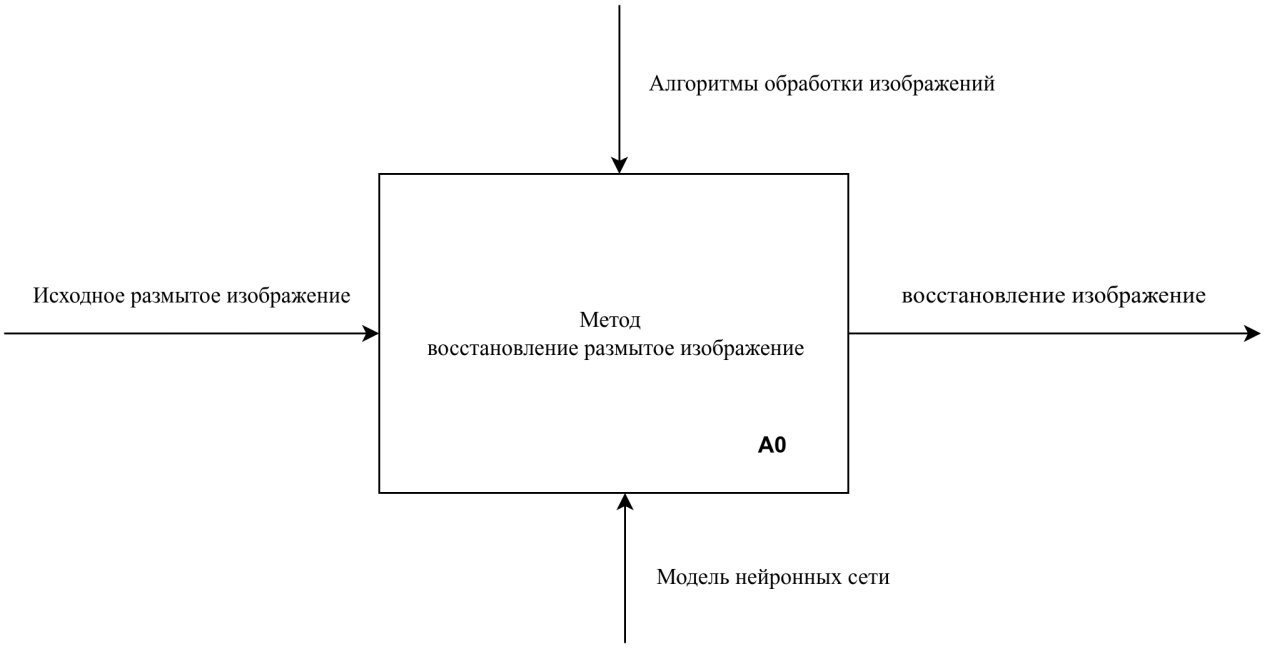

Рис. 1.1 – Формализация задачи восстановление размытие изображение нулевый уровень преобразований

На рисунке (1.2) представлено, формализация задачи восстановление размытие изображение первый уровень преобразований:

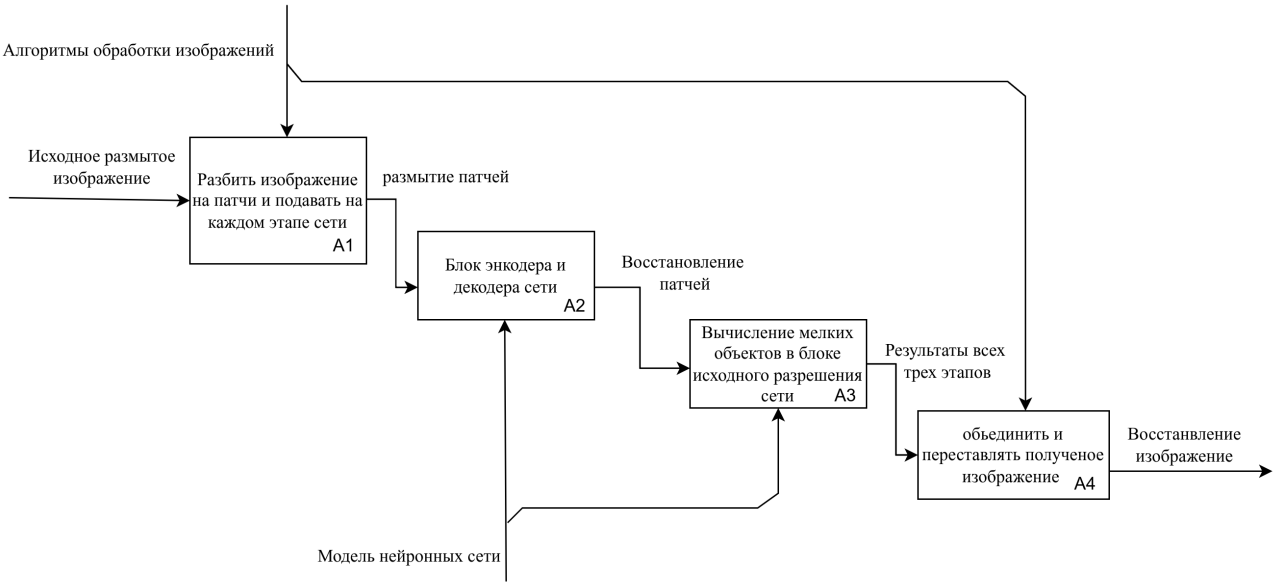

Рис. 1.2 – Формализация задачи восстановление размытие изображение первый уровень преобразований

1.2. Средства реализации

Выбор языка программирования

В качестве язык программирования выбран *Python* [6] по следующим причинам:

– *Python* обладает широким набором библиотек для реализации нейросетевых моделей;

– Существующие библиотеки позволяют ускорить обучение нейросетевых моделей за счёт параллельных вычислений и использования возможностей графического процессора.

Кроме того, язык программирования *Python* обеспечивает обширный спектр библиотек для работы с изображениями, что упрощает их обработку. Это становится необходимым при разделении изображения на несколько патчей и последующем объединении результатов после применения методов обработки к каждому патчу.

Выбор библиотеки для реализации нейросетевых моделей

Для реализации нейросетевой модели была выбрана библиотека *PyTorch* [7], которая обеспечивает возможность использования графического процессора

для вычислений с помощью средства параллельных вычислений *CUDA* [8]. Кроме того, данная библиотека предоставляет легкий интерфейс для реализации нейросетевых моделей.

Для работы с изображениями, такими как разделение на патчи и объединение результатов, была выбрана библиотека *Pillow* [9], которая предоставляет возможность работы с изображениями на языке программирования *Python*.

Аппаратные характеристики

Аппаратные характеристики машина на котором проводилось обучение сети такие:

- *ЦПУ*: Intel Core i7 8750H 2.2 ГГц [10];
- *Операционная система*: Ubuntu 20.04 Long Term Support [11];
- *Графический процессор*: GeForce RTX 1080 Ti [12];
- *Оперативная память*: 16 Гб.

1.3. Обучение нейросетевой модели

Перед обучением сети изображения из корпуса GoPro [13] были разделены на две выборки. Общее количество изображений в корпусе GoPro составляет 3206 пар. Изображения были разделены в соотношении 2:1, где 2103 пары были выбраны для обучения, а 1103 пары для валидации. Обучение сети проводилось в течение 7 часов и 43 минут на протяжении 100 эпох и количество параметров весов получилось 20127073. Кроме того, разрешение изображений в корпусе GoPro составляет 1280x720. Из-за низких аппаратных характеристик машины, на которой проводилось обучение сети, разрешение изображений было снижено до 480x360.

Скрипт на языке *Python* (листинг 1.1), для снижения разрешения изображений:

Листинг 1.1 – Скрипт для снижения разрешения изображений

```
1 def resize_image(input_path, output_path, target_width, target_height):
2     with Image.open(input_path) as img:
3         resized_img = img.resize((target_width, target_height))
4         resized_img.save(output_path)
5
```

```
dirs = {
    "./GoPro/test/input": "./GoPro/test/inputEdit",
    "./GoPro/test/target": "./GoPro/test/targetEdit",
    "./GoPro/train/input": "./GoPro/train/inputEdit",
    "./GoPro/train/target": "./GoPro/train/targetEdit",
    "./HIDE/test/input":  "./HIDE/test/inputEdit",
    "./HIDE/test/target": "./HIDE/test/targetEdit",
}

width, height = 480, 360
for input_dir, output_dir in dirs.items():
    for filename in os.listdir(input_dir):
        input_path = os.path.join(input_dir, filename)
        output_path = os.path.join(output_dir, filename)
        resize_image(input_path, output_path, width, height)
```

1.4. Функция потереи

Основной целью архитектуры MPRNet является вычисление потерь Шарбонье L_{char} и потерь края L_{edge}. Потеря Шарбонье вычисляется как попиксельная разница между предсказанным и восстановленным изображениями, а потеря края фокусируется на количественной оценке разницы в высокочастотных деталях между предсказанными и достоверными изображениями.

На листинге (1.4), предствалено реализации функцией потерей:

Листинг 1.2 – Реализации функции потерей

```
class CharbonnierLoss(nn.Module):
    def __init__(self, eps=1e-3):
        super(CharbonnierLoss, self).__init__()
        self.eps = eps

    def forward(self, x, y):
        diff = x - y
        loss = torch.mean(torch.sqrt((diff * diff) + (self.eps*self.eps)))
        return loss

class EdgeLoss(nn.Module):
    def __init__(self):
        super(EdgeLoss, self).__init__()
        k = torch.Tensor([[.05, .25, .4, .25, .05]])
        self.kernel = torch.matmul(k.t(),k).unsqueeze(0).repeat(3,1,1,1)
        if torch.cuda.is_available():
            self.kernel = self.kernel.cuda()
        self.loss = CharbonnierLoss()

```

```
20      def conv_gauss(self, img):
21          n_channels, _, kw, kh = self.kernel.shape
22          img = F.pad(img, (kw//2, kh//2, kw//2, kh//2), mode='replicate')
23          return F.conv2d(img, self.kernel, groups=n_channels)
24
25      def laplacian_kernel(self, current):
26          filtered    = self.conv_gauss(current)
27          down        = filtered[:,:,::2,::2]
28          new_filter  = torch.zeros_like(filtered)
29          new_filter[:,:,::2,::2] = down*4
30          filtered    = self.conv_gauss(new_filter)
31          diff = current - filtered
32          return diff
33
34      def forward(self, x, y):
35          loss = self.loss(self.laplacian_kernel(x), self.laplacian_kernel(y))
36          return loss
```

При обучении сети на каждой эпохе после того, как модель дает восстановленное изображение, производится вычисление потерь шарбонье и потерь края, которые затем суммируются как общая потеря модели.

На рисунке (1.3), представлены графики потерь Шарбонье и потерь края на каждой эпохе:

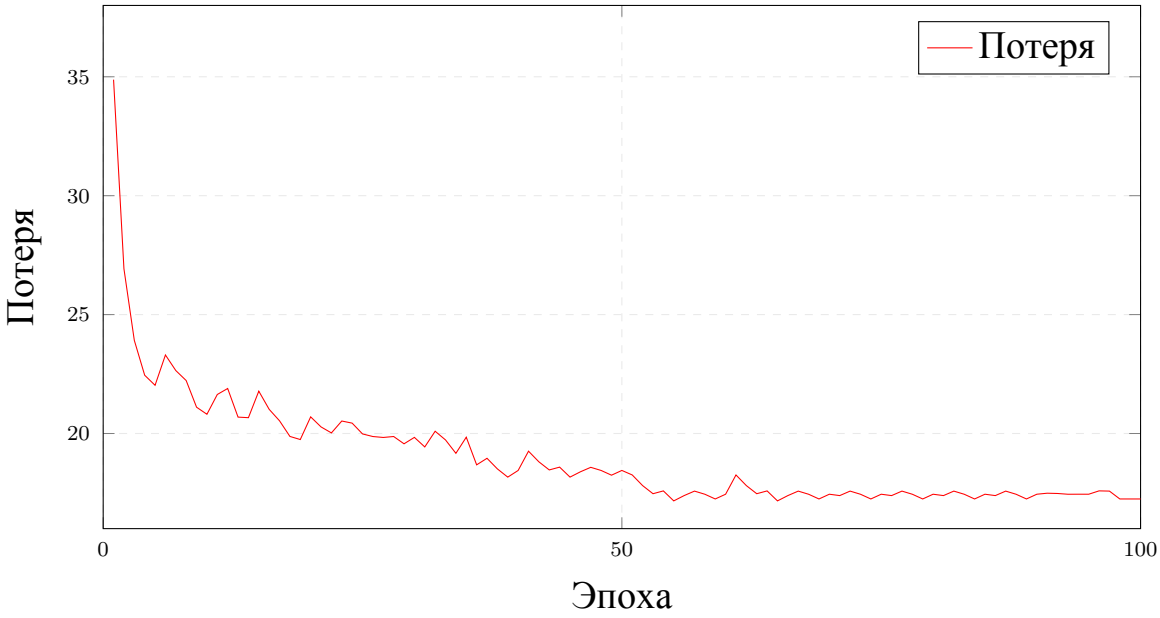

Рис. 1.3 – Потеря MPRNet на протяжении эпох

После каждых трёх эпох проводилась валидация сети, и вычислялось значение объективной метрики PSNR для полученного результата от сети.

На рисунке (1.4) представлены графики средних значений PSNR, получен-

ных после каждых трёх эпох проведения валидации сети.

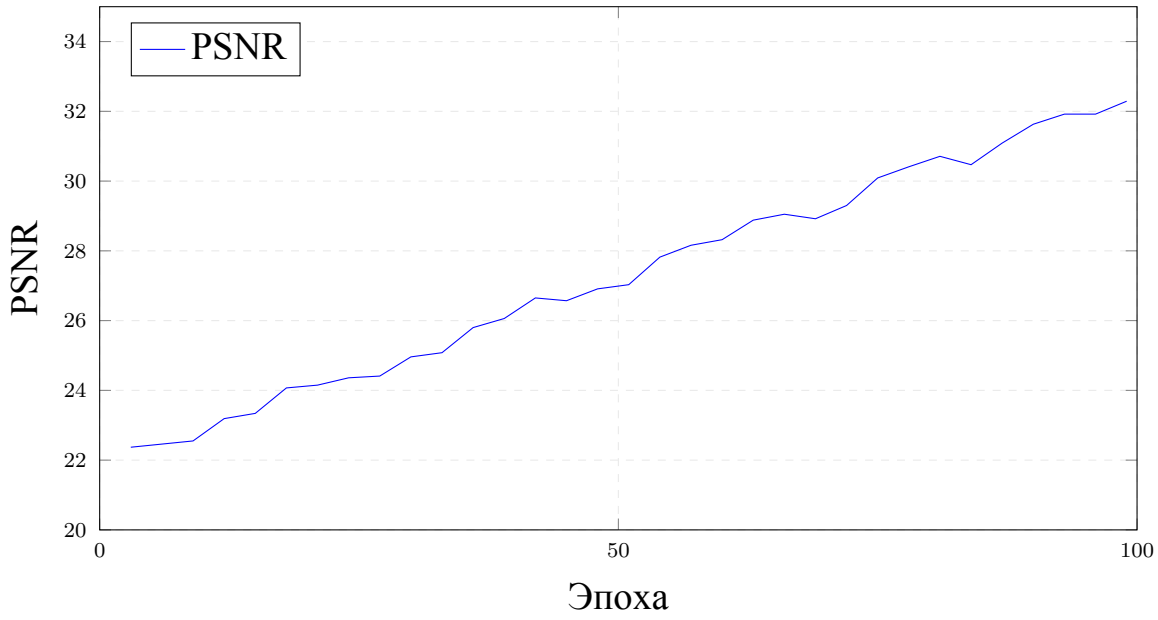

Рис. 1.4 – Средние значения PSNR, полученные MPRNet в процессе валидации сети

1.5. Пользовательский интерфейс

Так как программа будет работать в трех режимах обучении, валидации и оценке пользовательский интерфейс реализован в виде *CLI*, где пользователь перед запуском указывает желаемый режим работы программы и пути к входным и выходным данным.

На листинге (1.4), предствалено реализации интерфейса коммандной строки:

Листинг 1.3 – Реализация интерфейса коммандной строки часть 1

```
usage: main.py [-h] [--mode {train,predict}] [--input_dir INPUT_DIR]
    [--result_dir RESULT_DIR] [--weights WEIGHTS]
    [--dataset DATASET] [--gpus GPUS] [--config CONFIG]

Image Deblurring using MPRNet

optional arguments:
    -h, --help            show this help message and exit
    --mode {train,predict}
                Mode: train or predict
    --input_dir INPUT_DIR
                Directory of input images
    --result_dir RESULT_DIR
                Directory for results
```

```
15      --weights WEIGHTS     Path to weights
16      --dataset DATASET     Dataset for testing or validation
```

Листинг 1.4 – Реализация интерфейса коммандной строки часть 2

```
1       --gpus GPUS           CUDA_VISIBLE_DEVICES
2       --config CONFIG       Path to training configuration file
```

Также можно указывать параметры обучения через конфигурационный файл (*training.yml*).

На листинге (1.5), представлен содержимое конфигурационного файла:

Листинг 1.5 – Содержимое конфигурационного файла

```
1  GPU: [0,1,2,3]
2
3  OPTIM:
4      BATCH_SIZE: 16
5      NUM_EPOCHS: 200
6      LR_INITIAL: 2e-4
7      LR_MIN: 1e-6
8
9  TRAINING:
10     VAL_AFTER_EVERY: 5
11     RESUME: False
12     TRAIN_PS: 256
13     VAL_PS: 256
14     TRAIN_DIR: './Datasets/GoPro/train'
15     VAL_DIR: './Datasets/GoPro/test'
16     SAVE_DIR: './checkpoints'
```

1.6. Демонстрация работы программы

Для запуска программы в режиме оценки необходимо указать программе путь к входному изображению. Пользователь также должен указать путь к папке, где будут сохраняться результаты. Режим оценки модели проводится с использованием заранее обученных весов, а результатом является восстановленное изображение. На листинге 1.6 представлен пример работы программы.

Листинг 1.6 – Пример работы программы

```
1  $ python main.py --mode=predict --input_dir=testImgs --result_dir=testResult
2  Files saved at testResult.
```

На рисунке 1.5 представлен пример исходного и полученного изображений из сети.

Исходное изображение

Восстановленное изображение

Рис. 1.5 – Пример работы программы

1.7. Тестирование программное обеспечения

Как было ранее указано, одним из требований MPRNet является то, что входное изображение должно быть одинаково разделено на четыре части. Поэтому в режиме оценки любое изображение, подаваемое на вход, дополняется нулевыми пикселями. После того как сеть выдает результат, эти дополненные пиксели удаляются, и предоставляется разрешение, которое было подано на вход сети. Кроме того, можно проводить тестирование сети следующим образом:

1. После нескольких эпох обучения сети можно приостановить обучение на обучающей выборке и проводить тестирование сети на списке изображений валидационной выборки. Таким образом, можно оценить работу метода на этапе обучения;

2. С помощью обученных параметров весов можно проводить тестирование на некоторых парах изображений и оценить работу метода после прохождения этапа обучения. С использованием объективных метрик можно оценить работу метода после обучения;

3. Пользователь может сам выбрать одно или несколько изображений и провести оценку работы метода относительно выбранных изображений.

Вывод

В данном разделе проводилось описание выбранных средств реализации, аппаратных характеристик, процесса обучения нейросетевой модели и методов тестирования программного обеспечения.

Для реализации нейросетевой модели был выбран язык программирования Python, который обладает богатым набором библиотек для работы с нейросетями и обработки изображений, обучение нейросетевой модели проводилось на корпусе изображений GoPro, разрешение изображений было снижено для ускорения обучения.

Функция потерь модели включает в себя потерю Шарбонье и потерю края, которые оценивают разницу между предсказанными и исходными изображениями как по содержанию, так и по деталям.

Для тестирования программного обеспечения были разработаны методы, позволяющие оценить работу модели на различных этапах обучения, включая тестирование на валидационной выборке и отдельных изображениях.

2 Исследовательский раздел

2.1. Сравнение результатов, полученных методом, с результатами других методов

Для достижения данной цели были использованы обученные веса для сети, которые были получены в аналитическом разделе. Затем было проведено устранение размытия. Далее было вычислено значение метрики *PSNR* для оценки эффективности разработанного метода.

На рисунке 2.1 представлено изображение, которое было использовано для проведения оценки эффективности метода.

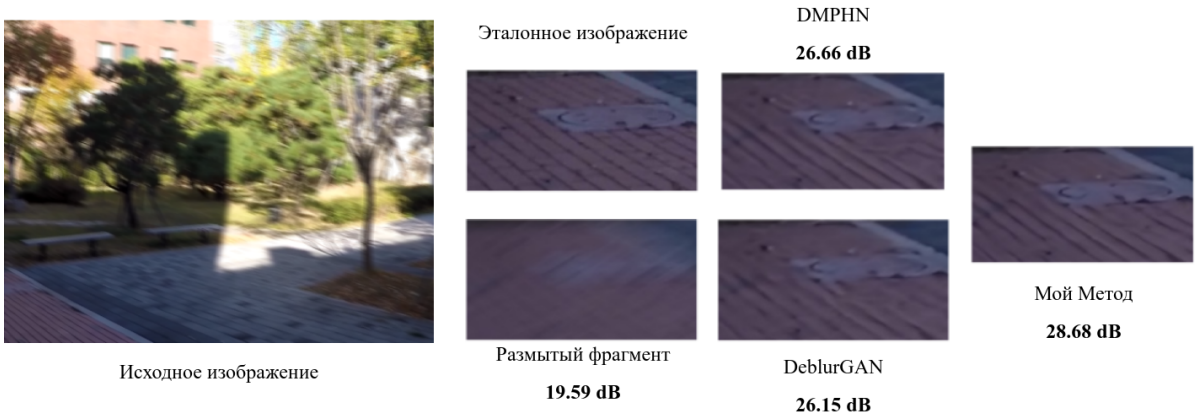

Рис. 2.1 – Изображение для оценки эффективности разработанного метода

В итоге полученное значение *PSNR* составило 28.68 децибела, что показывает значительное улучшение работы разработанного метода, в то время как для выбранное изображение *DMPHN* дает 26.66, а *DeblurGAN* - 26.15.

2.2. Проведение тестирования метода на корпусе данных

Для достижения данной цели был выбран еще один корпус данных *HIDE*, изображения которого не присутствовали в обучающем наборе данных, а также 1/3 часть изображений из корпуса данных *GoPro*, которые также не использовались при обучении сети. Для сравнения эффективности метода, аналогично предыдущему подразделу, было проведено устранение размытия с использованием обученных весов сети, после чего было выполнено сравнение метрик *PSNR* и *SSIM* относительно эталонного изображения и изображения, полученного из сети. Затем было рассмотрено среднее значение метрик *PSNR* и *SSIM*. В данном исследовании был проведен сравнительный анализ средних значений метрик

PSNR и *SSIM* относительно изображений, использованных для тестирования сети.

В таблице (2.1) представлен средние значения объективных метрик *PSNR* и *SSIM* относительно корпусов данных *GoPro* и *HIDE*.

Таблица 2.1 – Средние значения объективных метрик, полученные с использованием обученных весов сети, относительно корпусы данных *GoPro* и *HIDE*

	GoPro		HIDE	
Метрика / Метод	PSNR	SSIM	PSNR	SSIM
DeblurGAN	31.10	0.942	28.94	0.915
DMPHN	31.20	0.940	29.09	0.924
MPRNet	32.66	0.959	30.96	0.939

Как видно из полученных результатов метрик *PSNR* и *SSIM*, разработанный метод не только в деталях демонстрирует хорошие результаты, но также по значению объективных метрик превосходит другие методы и показывает наилучшие результаты.

2.3. Эффективность метода размытия на реальных изображениях

Для достижения этой цели было выбрано изображение из корпуса данных *RealBlur* [14]. Все изображения в этом корпусе сняты с помощью мобильных камер или фотоаппаратов. Разрешение этих изображений различно, и изображение делится на две категории:

1. *RealBlur-R*;
2. *RealBlur-J*.

Изображения в категории *RealBlur-R* создавались в автономном режиме путем применения к необработанным изображениям операций по балансу белого, демозакладке и уменьшению шума, а изображения в категории *RealBlur-J* формировались с помощью снятых изображений камеры в формате JPEG.

Было проведено тестирование на обеих этих категории, результат средние значение объективных метриков, полученного с помощью обучениеми весами сети на данные из корпуса данных *GoPro*, представлен на таблице 2.2.

Таблица 2.2 – Средние значения объективных метрик, полученные с использованием обученных весов сети, относительно корпуса данных *RealBlur*

	RealBlur-R		RealBlur-J	
Метрика / Метод	PSNR	SSIM	PSNR	SSIM
DeblurGAN	33.79	0.903	33.79	0.903
DMPHN	35.70	0.948	35.70	0.948
MPRNet	35.99	0.952	35.99	0.952

Исходя из полученных значений, можно сделать вывод, что метод хорошо справляется с изображениями из реального мира, поскольку корпус данных *RealBlur* содержит изображения, снятые с разных видов фотоаппаратов и разных разрешений. Однако, если обучить сеть на этих данных, то можно получить лучшие результаты.

Вывод

В данном разделе был проведен сравнительный анализ разработанного метода относительно других методов. Было проведено тестирование сети на реальных изображениях и сгенерированных размытых изображениях. В итоге были составлены таблицы с полученными результатами. По полученным данным можно сделать вывод, что метод хорошо справляется с изображениями, снятыми с разных видов фотоаппаратов и различных разрешений. Однако, чтобы гарантировать работу метода на различных видах изображений, лучшим способом является обучение сети на этих изображениях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Deep Semantic Face Deblurring / Z. Shen [и др.]. — 2018. — arXiv: 1803.03345 [cs.CV].
2. *Krishnan D.*, *Fergus R.* Fast Image Deconvolution using Hyper-Laplacian Priors // Advances in Neural Information Processing Systems. Т. 22 / под ред. Y. Bengio [и др.]. — Curran Associates, Inc., 2009. — Режим доступа, URL: https://proceedings.neurips.cc/paper_files/paper/2009/file/3dd48ab31d016ffcbf3314df2b3cb9ce-Paper.pdf.
3. *Lian Z.*, *Wang H.* An image deblurring method using improved U-Net model based on multilayer fusion and attention mechanism // Scientific Reports. — 2023. — Дек. — Т. 13, № 1. — С. 21402. — ISSN 2045-2322. — DOI: 10.1038/s41598-023-47768-4. — Режим доступа, URL: https://doi.org/10.1038/s41598-023-47768-4.
4. *Sun S. J.*, *Wu Q.*, *Li G. H.* Image Deblurring Algorithm for Overlap-Blurred Image // Advanced Measurement and Test X. Т. 439. — Trans Tech Publications Ltd, 10.2010. — С. 493—498. — (Key Engineering Materials). — DOI: 10.4028/www.scientific.net/KEM.439-440.493.
5. Deep Image Deblurring: A Survey / K. Zhang [и др.]. — 2022. — arXiv: 2201.10700 [cs.CV].
6. Python Programming Language. — Дата обращения: 02.05.2024. https://www.python.org/.
7. PyTorch: An Imperative Style, High-Performance Deep Learning Library / A. Paszke [и др.]. — 2019. — Дата обращения: 09.05.2024. https://pytorch.org/.
8. *Corporation N.* NVIDIA CUDA Software and GPU Parallel Computing Architecture. — 2011. — Дата обращения: 09.05.2024. https://developer.nvidia.com/cuda-zone.
9. *Clark F.* Python Imaging Library (PIL) / PythonWare. — 2002. — Режим доступа, URL: https://python-pillow.org/ ; Дата обращения: 09.05.2024.
10. Intel Core i7-8750H 2.2 GHz Processor. — Дата обращения: 10.05.2024. https://ark.intel.com/content/www/us/en/ark/products/134906/intel-core-i7-8750h-processor-9m-cache-up-to-4-10-ghz.html.

11. *Canonical Ltd.* Ubuntu 20.04 LTS. — 2020. — Дата обращения: 10.05.2024. https://releases.ubuntu.com/20.04/.

12. Nvidia GeForce GTX 1080 Ti video card. — Дата обращения: 10.05.2024. https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti/.

13. *Nah S.*, *Hyun Kim T.*, *Mu Lee K.* Deep multi-scale convolutional neural network for dynamic scene deblurring // Proceedings of the IEEE conference on computer vision and pattern recognition. — 2017. — С. 3883—3891.

14. Real-world blur dataset for learning and benchmarking deblurring algorithms / J. Rim [и др.] // Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV 16. — Springer. 2020. — С. 184—201.

ПРИЛОЖЕНИЕ А

Реализации архитектура сети

Листинг 2.1 – Сверточный модуль/блок внимания (CAB)

```
class CAB(nn.Module):
    def __init__(self, n_feat, kernel_size, reduction, bias, act):
        super(CAB, self).__init__()
        modules_body = []
        modules_body.append(conv(n_feat, n_feat, kernel_size, bias=bias))
        modules_body.append(act)
        modules_body.append(conv(n_feat, n_feat, kernel_size, bias=bias))

        self.CA = CALayer(n_feat, reduction, bias=bias)
        self.body = nn.Sequential(*modules_body)

    def forward(self, x):
        res = self.body(x)
        res = self.CA(res)
        res += x
        return res
```

Листинг 2.2 – Модуль внимания канала (CAB)

```
class CALayer(nn.Module):
    def __init__(self, channel, reduction=16, bias=False):
        super(CALayer, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.conv_du = nn.Sequential(
            nn.Conv2d(channel, channel // reduction, 1,
                      padding=0, bias=bias),
            nn.ReLU(inplace=True),
            nn.Conv2d(channel // reduction, channel, 1,
                      padding=0, bias=bias),
            nn.Sigmoid()
        )

    def forward(self, x):
        y = self.avg_pool(x)
        y = self.conv_du(y)
        return x * y
```

Листинг 2.3 – Подсеть исходного разрешения (ORSNet) часть 1

```
class ORSNet(nn.Module):
    def __init__(self, n_feat, scale_orsnetfeats, kernel_size,
        reduction, act, bias, scale_unetfeats, num_cab):
        super(ORSNet, self).__init__()

        self.orb1 =
            ORB(n_feat+scale_orsnetfeats, kernel_size,
                reduction, act, bias, num_cab)
        self.orb2 =
            ORB(n_feat+scale_orsnetfeats, kernel_size,
                reduction, act, bias, num_cab)
        self.orb3 =
            ORB(n_feat+scale_orsnetfeats, kernel_size,
                reduction, act, bias, num_cab)

        self.up_enc1 = UpSample(n_feat, scale_unetfeats)
        self.up_dec1 = UpSample(n_feat, scale_unetfeats)

        self.up_enc2 =
            nn.Sequential(UpSample(n_feat+scale_unetfeats, scale_unetfeats),
                UpSample(n_feat, scale_unetfeats))
        self.up_dec2 =
            nn.Sequential(UpSample(n_feat+scale_unetfeats, scale_unetfeats),
                UpSample(n_feat, scale_unetfeats))

        self.conv_enc1 =
            nn.Conv2d(n_feat, n_feat+scale_orsnetfeats,
                kernel_size=1, bias=bias)
        self.conv_enc2 =
            nn.Conv2d(n_feat, n_feat+scale_orsnetfeats,
                kernel_size=1, bias=bias)
        self.conv_enc3 =
            nn.Conv2d(n_feat, n_feat+scale_orsnetfeats,
                kernel_size=1, bias=bias)

        self.conv_dec1 =
            nn.Conv2d(n_feat, n_feat+scale_orsnetfeats,
                kernel_size=1, bias=bias)
        self.conv_dec2 =
            nn.Conv2d(n_feat, n_feat+scale_orsnetfeats,
                kernel_size=1, bias=bias)
        self.conv_dec3 =
            nn.Conv2d(n_feat, n_feat+scale_orsnetfeats,
                kernel_size=1, bias=bias)
```

Листинг 2.4 – Подсеть исходного разрешения (ORSNet) часть 2

```
    def forward(self, x, encoder_outs, decoder_outs):
        x = self.orb1(x)
        x = x + self.conv_enc1(encoder_outs[0]) +
            self.conv_dec1(decoder_outs[0])

        x = self.orb2(x)
        x = x + self.conv_enc2(self.up_enc1(encoder_outs[1])) +
            self.conv_dec2(self.up_dec1(decoder_outs[1]))

        x = self.orb3(x)
        x = x + self.conv_enc3(self.up_enc2(encoder_outs[2])) +
            self.conv_dec3(self.up_dec2(decoder_outs[2]))

        return x
```

Листинг 2.5 – Кодировщик UNet часть 1

```
class Encoder(nn.Module):
    def __init__(self, n_feat, kernel_size, reduction, act,
        bias, scale_unetfeats, csff):
        super(Encoder, self).__init__()

        self.encoder_level1 = [CAB(n_feat,                      kernel_size,
            reduction, bias=bias, act=act) for _ in range(2)]
        self.encoder_level2 = [CAB(n_feat+scale_unetfeats,      kernel_size,
            reduction, bias=bias, act=act) for _ in range(2)]
        self.encoder_level3 = [CAB(n_feat+(scale_unetfeats*2), kernel_size,
            reduction, bias=bias, act=act) for _ in range(2)]

        self.encoder_level1 = nn.Sequential(*self.encoder_level1)
        self.encoder_level2 = nn.Sequential(*self.encoder_level2)
        self.encoder_level3 = nn.Sequential(*self.encoder_level3)

        self.down12  = DownSample(n_feat, scale_unetfeats)
        self.down23  = DownSample(n_feat+scale_unetfeats, scale_unetfeats)

        if csff:
            self.csff_enc1 = nn.Conv2d(n_feat,
                n_feat,                      kernel_size=1, bias=bias)
            self.csff_enc2 = nn.Conv2d(n_feat+scale_unetfeats,
                n_feat+scale_unetfeats,      kernel_size=1, bias=bias)
            self.csff_enc3 = nn.Conv2d(n_feat+(scale_unetfeats*2),
                n_feat+(scale_unetfeats*2), kernel_size=1, bias=bias)

            self.csff_dec1 = nn.Conv2d(n_feat,
                n_feat,                      kernel_size=1, bias=bias)
```

Листинг 2.6 – Кодировщик UNet часть 2

```
            self.csff_dec2 = nn.Conv2d(n_feat+scale_unetfeats,
                n_feat+scale_unetfeats,      kernel_size=1, bias=bias)
            self.csff_dec3 = nn.Conv2d(n_feat+(scale_unetfeats*2),
                n_feat+(scale_unetfeats*2), kernel_size=1, bias=bias)

    def forward(self, x, encoder_outs=None, decoder_outs=None):
        enc1 = self.encoder_level1(x)
        if (encoder_outs is not None) and (decoder_outs is not None):
            enc1 = enc1 + self.csff_enc1(encoder_outs[0]) +
                self.csff_dec1(decoder_outs[0])

        x = self.down12(enc1)

        enc2 = self.encoder_level2(x)
        if (encoder_outs is not None) and (decoder_outs is not None):
            enc2 = enc2 + self.csff_enc2(encoder_outs[1]) +
                self.csff_dec2(decoder_outs[1])

        x = self.down23(enc2)

        enc3 = self.encoder_level3(x)
        if (encoder_outs is not None) and (decoder_outs is not None):
            enc3 = enc3 + self.csff_enc3(encoder_outs[2]) +
                self.csff_dec3(decoder_outs[2])

        return [enc1, enc2, enc3]
```

Листинг 2.7 – Декодер UNet часть 1

```
class Decoder(nn.Module):
    def __init__(self, n_feat, kernel_size, reduction, act,
        bias, scale_unetfeats):
        super(Decoder, self).__init__()

        self.decoder_level1 = [CAB(n_feat,                      kernel_size,
            reduction, bias=bias, act=act) for _ in range(2)]
        self.decoder_level2 = [CAB(n_feat+scale_unetfeats,      kernel_size,
            reduction, bias=bias, act=act) for _ in range(2)]
        self.decoder_level3 = [CAB(n_feat+(scale_unetfeats*2), kernel_size,
            reduction, bias=bias, act=act) for _ in range(2)]

        self.decoder_level1 = nn.Sequential(*self.decoder_level1)
        self.decoder_level2 = nn.Sequential(*self.decoder_level2)
        self.decoder_level3 = nn.Sequential(*self.decoder_level3)

        self.skip_attn1 = CAB(n_feat,                  kernel_size,
            reduction, bias=bias, act=act)
```

Листинг 2.8 – Декодер UNet часть 2

```
        self.skip_attn2 = CAB(n_feat+scale_unetfeats, kernel_size,
            reduction, bias=bias, act=act)

        self.up21  = SkipUpSample(n_feat, scale_unetfeats)
        self.up32  = SkipUpSample(n_feat+scale_unetfeats, scale_unetfeats)

    def forward(self, outs):
        enc1, enc2, enc3 = outs
        dec3 = self.decoder_level3(enc3)

        x = self.up32(dec3, self.skip_attn2(enc2))
        dec2 = self.decoder_level2(x)

        x = self.up21(dec2, self.skip_attn1(enc1))
        dec1 = self.decoder_level1(x)

        return [dec1,dec2,dec3]
```

Листинг 2.9 – Сеть многоступенчатой прогрессивной реставрации изображения (MPRNet) часть 1

```
class MPRNet(nn.Module):
    def __init__(self, in_c=3, out_c=3, n_feat=96, scale_unetfeats=48,
                 scale_orsnetfeats=32, num_cab=8, kernel_size=3,
                 reduction=4, bias=False):
        super(MPRNet, self).__init__()

        act=nn.PReLU()
        self.shallow_feat1 = nn.Sequential(conv(in_c, n_feat,
            kernel_size, bias=bias),
            CAB(n_feat,kernel_size, reduction, bias=bias, act=act))
        self.shallow_feat2 = nn.Sequential(conv(in_c, n_feat,
            kernel_size, bias=bias),
            CAB(n_feat,kernel_size, reduction, bias=bias, act=act))
        self.shallow_feat3 = nn.Sequential(conv(in_c, n_feat,
            kernel_size, bias=bias),
            CAB(n_feat,kernel_size, reduction, bias=bias, act=act))

        self.stage1_encoder = Encoder(n_feat, kernel_size, reduction,
            act, bias, scale_unetfeats, csff=False)
        self.stage1_decoder = Decoder(n_feat, kernel_size, reduction,
            act, bias, scale_unetfeats)

        self.stage2_encoder = Encoder(n_feat, kernel_size, reduction,
            act, bias, scale_unetfeats, csff=True)
        self.stage2_decoder = Decoder(n_feat, kernel_size, reduction,
            act, bias, scale_unetfeats)
```

Листинг 2.10 – Сеть многоступенчатой прогрессивной реставрации изображения (MPRNet) часть 2

```
        self.stage3_orsnet = ORSNet(n_feat, scale_orsnetfeats,
            kernel_size, reduction, act, bias, scale_unetfeats, num_cab)

        self.sam12 = SAM(n_feat, kernel_size=1, bias=bias)
        self.sam23 = SAM(n_feat, kernel_size=1, bias=bias)

        self.concat12  = conv(n_feat*2, n_feat, kernel_size, bias=bias)
        self.concat23  = conv(n_feat*2, n_feat+scale_orsnetfeats,
            kernel_size, bias=bias)
        self.tail     = conv(n_feat+scale_orsnetfeats,
            out_c, kernel_size, bias=bias)

    def forward(self, x3_img):
        H = x3_img.size(2)
        W = x3_img.size(3)

        x2top_img  = x3_img[:,:,0:int(H/2),:]
        x2bot_img  = x3_img[:,:,int(H/2):H,:]

        x1ltop_img = x2top_img[:,:,:,0:int(W/2)]
        x1rtop_img = x2top_img[:,:,:,int(W/2):W]
        x1lbot_img = x2bot_img[:,:,:,0:int(W/2)]
        x1rbot_img = x2bot_img[:,:,:,int(W/2):W]

        x1ltop = self.shallow_feat1(x1ltop_img)
        x1rtop = self.shallow_feat1(x1rtop_img)
        x1lbot = self.shallow_feat1(x1lbot_img)
        x1rbot = self.shallow_feat1(x1rbot_img)

        feat1_ltop = self.stage1_encoder(x1ltop)
        feat1_rtop = self.stage1_encoder(x1rtop)
        feat1_lbot = self.stage1_encoder(x1lbot)
        feat1_rbot = self.stage1_encoder(x1rbot)

        feat1_top = [torch.cat((k,v), 3) for k,v in
            zip(feat1_ltop,feat1_rtop)]
        feat1_bot = [torch.cat((k,v), 3) for k,v in
            zip(feat1_lbot,feat1_rbot)]

        res1_top = self.stage1_decoder(feat1_top)
        res1_bot = self.stage1_decoder(feat1_bot)

        x2top_samfeats, stage1_img_top = self.sam12(res1_top[0], x2top_img)
        x2bot_samfeats, stage1_img_bot = self.sam12(res1_bot[0], x2bot_img)
```

Листинг 2.11 – Сеть многоступенчатой прогрессивной реставрации изображения (MPRNet) часть 3

```
        stage1_img = torch.cat([stage1_img_top, stage1_img_bot],2)
        x2top  = self.shallow_feat2(x2top_img)
        x2bot  = self.shallow_feat2(x2bot_img)

        x2top_cat = self.concat12(torch.cat([x2top, x2top_samfeats], 1))
        x2bot_cat = self.concat12(torch.cat([x2bot, x2bot_samfeats], 1))

        feat2_top = self.stage2_encoder(x2top_cat, feat1_top, res1_top)
        feat2_bot = self.stage2_encoder(x2bot_cat, feat1_bot, res1_bot)

        feat2 = [torch.cat((k,v), 2) for k,v in zip(feat2_top,feat2_bot)]

        res2 = self.stage2_decoder(feat2)

        x3_samfeats, stage2_img = self.sam23(res2[0], x3_img)

        x3     = self.shallow_feat3(x3_img)

        x3_cat = self.concat23(torch.cat([x3, x3_samfeats], 1))

        x3_cat = self.stage3_orsnet(x3_cat, feat2, res2)

        stage3_img = self.tail(x3_cat)

        return [stage3_img+x3_img, stage2_img, stage1_img]
```