УДК 004.8

Быстрая сверточная нейронная сеть сверхвысокого разрешения изображения

Черкашин Александр Михайлович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс использования модели нейронный сети для улучшения качества изображения с увеличением разрешения. В работе использовалась библиотека Torch, модель, и представленный набор данных. В результате работы получены метрика и оценка модели для улучшения качества изображения с увеличением разрешения.
Ключевые слова: FSRCNN, Сверточные нейронные сети, Сверхвысокого разрешения, Torch.

Creating the effect of moving stars in the shader language

Cherkashin Alexander Mihailovich
Sholom-Aleichem Priamursky State University
student

Abstract
This article describes the process of using a neural network model to improve image quality with increasing resolution. The Torch library, the model, and the presented data set were used in the work. As a result of the work, a metric and an estimate of the model were obtained to improve the image quality with increasing resolution.
Keywords: FSRCNN, Convolutional Neural Network, Super-Resolution, Torch.

1 Введение

1.1 Актуальность исследования

Актуальность исследования заключается в ее потенциале для значительного улучшения задач обработки изображений и видео, таких как улучшение изображений с низким разрешением. Предлагаемая модель также может быть использована в таких отраслях, как развлечения, фотография и игры.

1.2 Цель исследования

Целью работы является создания и обучение модели для увеличения разрешения изображения с улучшением качества.

1.3 Обзор исследований

К. Жанг, Щ. Зуо, Л. Жанг рассматривает единую сверточную сеть для устранения множественных деградаций, они смогли достичь превосходных результатов по сравнению с традиционными методами супер-разрешения. Эксперименты, проведенные на различных наборах данных, показывают, что предлагаемый их подход не только дает высококачественные результаты, но и значительно сокращает время вычислений. В целом, это исследование очень важно для области обработки изображений и представляет многообещающее направление для будущих исследований [1].

Л. С. Пассарелла и др. отмечает разработку быстрого нейронной сети (FSRCNN) для реконструкции данных моделирования системы Земли с высоким разрешением (ESM). В исследовании представлены многообещающие результаты, что указывает на то, что предлагаемый метод FSRCNN превосходит традиционные методы интерполяции с точки зрения точности и времени вычисления. Применение исследователей в области глубокого обучения в области моделирования и моделирования климата повышает точность и качество данных ESM, что имеет решающее значение для развития нашего понимания сложных систем Земли. В целом, это исследование полезно в разработке передовых методов обработки данных, которые могут помочь преодолеть разрыв между разрешением данных ESM и масштабом природных явлений [2].

С. Парк и др. предлагает в области здравоохранения и медицины подход к супер-разрешению изображений магнитно-резонансных (МРТ) с использованием метода сверточной нейронной сети, основанного на автоэкодере. Техника генерирует изображения с более высоким разрешением от МРТ с низким разрешением, что может значительно повысить диагностическую точность процедур медицинской визуализации. В статье подробно описывается реализация и производительность предлагаемого метода с помощью различных экспериментов, показывая многообещающие результаты с точки зрения улучшения качества изображения. В целом, исследование дает ценную информацию о разработке передовых методов визуализации МРТ с потенциальными клиническими применениями в будущем [3].

Ц. Донг, Ц. Ц. Лоы, Х. Танг провели исследование по повышению эффективности сверточной нейронной сети супер разрешения. В их исследовании используется инновационный метод использования компактной структуры сети для минимизации вычислительных расходов при сохранении высоких уровней точности. Результаты демонстрируют значительные достижения как в скорости, так и в качестве улучшения изображения. В целом, это исследование является важным вкладом в область компьютерного зрения и, несомненно, повлияет на будущие разработки в области обработки изображений с высоким разрешением [4].

Й. Чэнь и др. исследуют сверхрезок МРТ мозга с использованием 3D глубоко плотно связанных нейронных сетей представляют многообещающие результаты для улучшения разрешения и качества изображений МРТ. В

исследовании использовались большой набор данных и передовые методы машинного обучения для улучшения деталей и ясности изображений мозга. Использование исследователей в 3D глубоко плотно связанных нейронных сетях представляет собой новый подход к супер разрешению изображения, и их результаты могут иметь значительные последствия как для клинических, так и для исследований. В целом, это исследование дает ценную информацию о потенциале методов глубокого обучения для улучшения медицинской визуализации и создает основу для будущих достижений в этой области [5].

В. Ли и др. Рассматривает задачу использования информации для супер-разрешения изображения, что является важным аспектом анализа изображений. Авторы разработали метод, который использует игровое обучение для обучения сети глубокого обучения для повышения производительности в супер-разрешении изображения. Результаты исследования показывают, что предложенный метод повышает точность и скорость супер-разрешения изображения. В целом, исследование предоставляет ценную информацию, которая будет способствовать разработке передовых методов анализа изображений в будущем [6].

В. Мане, С. Ядхав, П. Лал описывают сверхрезолирование MRI с использованием 3D более быстрого супер-разрешения архитектуры нейронной сети которое дает новое представление о разработке передовых методов обработки изображений для медицинской визуализации. Исследование показывает многообещающие результаты в суперразрешающих МРТ с низким разрешением, которые могут повысить точность медицинской диагностики и лечения. Использование алгоритмов глубокого обучения и нейронных сетей также предлагает новый подход, который потенциально может снизить зависимость от человеческого опыта и ручных вмешательств в обработке изображений. В целом, это исследование имеет значительные последствия для продвижения медицинской визуализации и потенциально может внести ценный вклад в область диагностики и лечения [7].

Л. С. Пассарелла и др. исследуют использование быстрой сверточной нейронной сети (FSRCNN) в улучшении разрешения климатических данных. Исследование показало, что FSRCNN является многообещающим подходом для улучшения разрешения климатических данных, поскольку он не только быстрее, чем традиционные методы интерполяции, но и обеспечивает более высокие результаты качества. В целом, это хорошо выполненное исследование, которое дает ценную информацию для исследователей в области климатической науки [8].

С. М. Уорд и др. исследуют оценку качества изображения как средство определения эффективности и ограничений сверточной нейронной сети супер-разрешения (SRCNN). Они углубляются в различные показатели оценки качества изображения и оценивают эффективность SRCNN по сравнению с традиционными методами интерполяции. Результаты их исследования демонстрируют потенциал SRCNN в качестве многообещающего подхода для улучшения разрешения изображений, подчеркивая его превосходную производительность с точки зрения визуального качества и объективных

оценок. В целом, это исследование дает ценную информацию о возможностях SRCNN и его потенциальных приложениях в области обработки изображений [9].

Х. Рен, М. Эль-Хами, Й. Ли используют глубокие сверточные нейронные сети, команда разработала каскадную обученную и обрезанную сеть, которая способна точно улучшить разрешение изображения. Их исследование представляет собой убедительный аргумент в пользу эффективности CT-SRCNN и дополнительно демонстрирует его способность превосходить существующие методы супер-разрешения. Это исследование предлагает ценный вклад в область компьютерного зрения и обработки изображений и имеет перспективы для будущих приложений в ряде отраслей [10].

С. К. Вб. предлагает метод включает использование глубокой нейронной сети, обученной на большом наборе данных изображений с низким разрешением, для создания версий этих изображений с высоким разрешением. Авторы оценивают эффективность своего подхода, используя различные показатели качества изображения, и сравнивают его с другими современными методами. Результаты показывают, что подход SRCNN очень эффективен для улучшения качества восприятия изображений с низким разрешением [11].

Н. Абураед. используют алгоритмы глубокого обучения для повышения разрешения и четкости изображения в гипертральных изображениях дистанционного зондирования, которые имеют явное применение в области наблюдения за Землей и спутниковой съемки. Результаты показывают заметное улучшение качества изображения, измеряемое пиковым отношением сигнал-шум и индексом структурного подобия. В нем подчеркивается потенциал подходов, основанных на глубоком обучении, для улучшения обработки изображений и анализа данных в областях дистанционного зондирования [12].

С. Янг, Х. Щанг сосредотачиваются на разработке метода объединения пространственно-временной информации из данных мультисенсорного дистанционного зондирования с использованием разреженного представления и сверточной нейронной сети сверхвысокого разрешения (SRCNN). Этот тип исследований имеет потенциальное применение в таких областях, как мониторинг окружающей среды и реагирование на стихийные бедствия. В целом, исследователи в области дистанционного зондирования могут найти это исследование актуальным и информативным [13].

К. Праджапати и др. предлагает новый подход к суперразрешению тепловых изображений с использованием сверточной нейронной сети с разделением каналов (Chasnet). Результаты их экспериментов показывают, что Chasnet превосходит другие современные методы сверхвысокого разрешения, достигая лучшего качества изображения с точки зрения визуального восприятия и объективных показателей оценки. В исследовании дается подробное описание предлагаемого метода, включая архитектуру сети, функцию потерь и процедуру обучения. В целом, исследование представляет собой важный вклад в область сверхвысокого разрешения тепловизионных

изображений с потенциальными приложениями в различных областях, таких как наблюдение, медицинская визуализация и промышленный контроль [14].

Б. З. Демираы, М. Сит, И. Демир предлагают такой подход на основе глубоких сверточных нейронных сетей для эффективного сверхразрешения изображений. Подход включает использование состязательного обучения для создания более реалистичного изображения с высоким разрешением из изображения с низким разрешением. В исследовании утверждается, что оно дает лучшие результаты, чем современные методы сверхвысокого разрешения [15].

Ы. Зоу представляет собой реконструкции инфракрасных изображений со сверхвысоким разрешением с использованием сверточной нейронной сети с пропуском соединений. Авторы демонстрируют свой подход на смоделированных и реальных инфракрасных изображениях и сообщают о значительном улучшении качества изображения по сравнению с традиционными методами сверхвысокого разрешения. Использование пропускных соединений позволяет сети включать информацию из слоев с более низким разрешением, что позволяет ей лучше восстанавливать мелкие детали. В целом, исследование представляет собой многообещающий подход к улучшению разрешения инфракрасных изображений, который может найти применение в таких областях, как дистанционное зондирование и медицинская визуализация [16].

2 Рабочий процесс

2.1. Набор данных

Исходные данные взяты сайта wallhaven.cc. В исходных данных представлено серия изображения высокого разрешения случайно подобранных любых тем и любых размеров.

Мы загружали изображения количеством 512 в программу, и количество копий изображений - 100, количество одновременно для обучающий серии изображения (batch_size) - 16 (рис 1).

Каждое загруженное изображение обрабатывается поэтапно:

1. Обрезается с размером на 512x512 пикселей случайного расположения.

2. Уменьшается до 80x80 пикселей.

3. Случайно поворачивается на максимум 360 градусов.

4. Преобразуется цветовая модель в YCrCb и в однокомпонентный цвет в Y.

5. Разделяется на два части, одно изображение - полный размер (названия hr), а второе - уменьшенный размер в 2 раза (название lr).

6. Преобразуется в тензор.

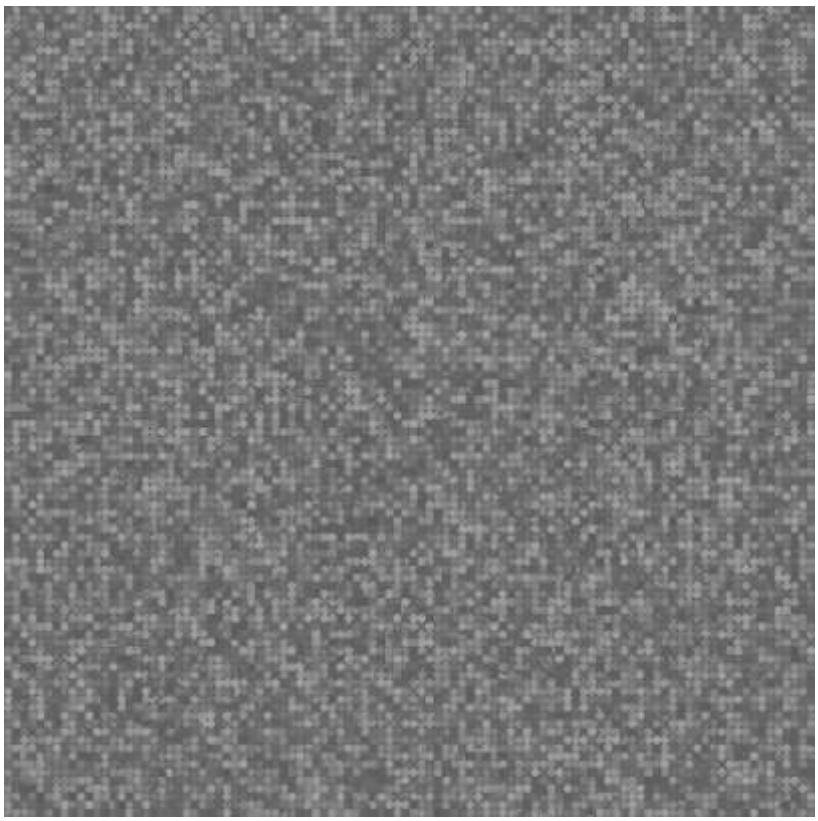

Рисунок 1. Обработанный набор данных, представленные по количеству 10000, по размеру 40x40

Рисунок 2. Результат загрузки и обработки набора данных

Количество загруженных изображения 51200 на каждый класс lr, hr. Всего 102400. Размер изображения lr — 40x40, hr — 80x80.

Листинг 2.1. Набор действия для обработки набор данных

```
self.resize_image = 512
self.transform = [
        Trans.Compose([
                transforms.RandomCrop(self.resize_image),
                transforms.Resize(self.size_image * self.upscale_factor),
                transforms.RandomRotation(360),
                Trans.ColorConvert("rgb", "ycrcb", "cv2"),
                transforms.Grayscale(num_output_channels=1),
                [Trans.Empty(), transforms.Resize(self.size_image)],
```

```
10                  transforms.ToTensor()
11              ])
12  ]
```

Строка 1 - размер изображения

Строка 2 — 11. Список действия для обработки набор данных.

2.1. Модель

В данной статье описывается модель FSRCNN (fast super resolution convolutional neural network, быстрая сверточная нейронная сеть со сверхвысоким разрешением), с краткими пересказами из автора статьи Ц. Донг, Ц. Ц. Лоы, Х Танг [4].

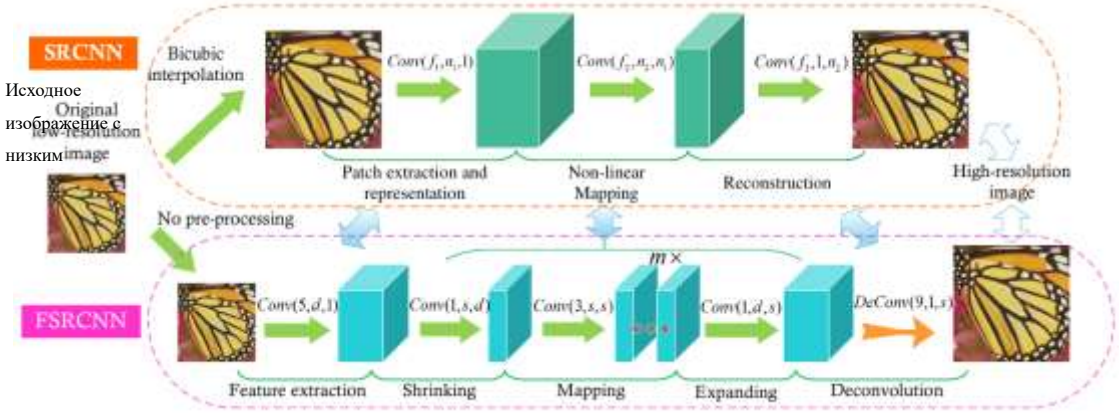

Рисунок 3. Различие SRCNN от FSRCNN

В отличие SRCNN (super resolution convolutional neural network, сверточная нейронная сеть сверхвысокого разрешения) от FSRCNN (fast super resolution convolutional neural network, быстрая сверточная нейронная сеть со сверхвысоким разрешением) FSRCNN принимает исходные изображения с низким разрешение, в качестве входных данных без бикубической интерполяции. В конце сети слоя вводится обратная свертка (Deconvolution) для повышения частоты дискретизации.

В второй этап, нелинейное отображение в SRCNN заменяется тремя шагами в FSRCNN, а именно, этапом сжатия, отображения и расширения.

В третий этап FSRCNN использует фильтры меньшего размера и более глубокую сеть (рис 2).

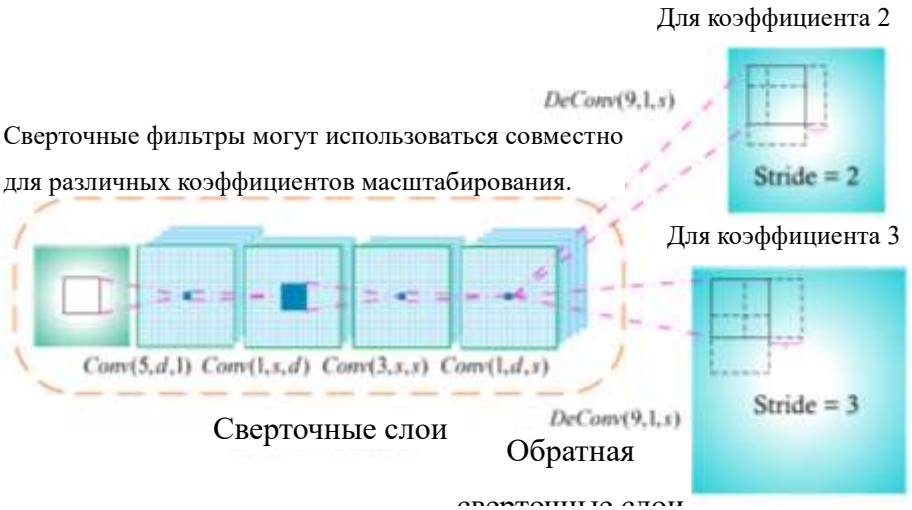

Рисунок 4. Структура слоя FSRCNN для различных коэффициента масштабирования

Листинг 2.2. Структура модели FSRCNN

```
FSRCNN(
        (feature_extraction): Sequential(
                (0): Conv2d(1, 56, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
                (1): PReLU(num_parameters=56)
        )
        (shrink): Sequential(
                (0): Conv2d(56, 12, kernel_size=(1, 1), stride=(1, 1))
                (1): PReLU(num_parameters=12)
        )
        (map): Sequential(
                (0): Conv2d(12, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (1): PReLU(num_parameters=12)
                (2): Conv2d(12, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (3): PReLU(num_parameters=12)
                (4): Conv2d(12, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (5): PReLU(num_parameters=12)
                (6): Conv2d(12, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (7): PReLU(num_parameters=12)
        )
        (expand): Sequential(
                (0): Conv2d(12, 56, kernel_size=(1, 1), stride=(1, 1))
                (1): PReLU(num_parameters=56)
        )
        (deconv): ConvTranspose2d(56, 1, kernel_size=(9, 9), stride=(2, 2), padding=(4, 4), output_padding=(1, 1))
)
```

Модель использована в программе [17].

Feature extraction: это часть похожа на первую часть SRCNN, но FSRCNN выполняет извлечение признаков исходного изображение LR без интерполяции. Размер фильтра 5 x 5.

shrink: На этом этапе извлекаются признаки LR и затем сокращается количество параметров ради сокращение вычислительной сложности. Размер ядра фильтра 1 x 1 использовали слой для уменьшение вычислительные сложности.

Non-linear mapping (Map): Нелинейное отображение, слой размер фильтра 5 x 5 дает лучшее качество чем размер 1 x 1, однако соотношение производительность и качество выбрали размер 3 x 3. Чтобы улучшить

качество изображение использовали несколько слоев, чтобы увеличить точность и сложность отображения (рис 3).

Expanding (expand): Расширяющийся слой действует как обратный сжимающемуся слою.

Deconvolution: Слой (обратная свертка) повышает дискретизацию и агрегирует данные и объединяет предыдущие функции с помощью набора фильтров обратный свертки. В результате обработки слоя на выходе получается размер изображение k раз больше чем входное и исходное размер изображения.

PReLU: Функция активация после каждый слоя свертки, используется чтобы избежать «мертвые признаков» (затухающий градиента) полученных нулевыми градиентами в ReLU. Сеть использующийся активатора PReLU более стабильные чем сети активатора ReLU.

Листинг 2.3. Параметры для обучения модели

```
1  self.lr = 1e-3
2  self.upscale_factor = 2
3  self.model = {
4          "fsrcnn": FSRCNN(self.upscale_factor).to(self.device)
5  }
6  self.optimizer = {
7          "fsrcnn": optim.SGD([{"params": self.model["fsrcnn"].feature_extraction.parameters()},
8                               {"params": self.model["fsrcnn"].shrink.parameters()},
9                               {"params": self.model["fsrcnn"].map.parameters()},
10                              {"params": self.model["fsrcnn"].expand.parameters()},
11                              {"params": self.model["fsrcnn"].deconv.parameters(), "lr": self.lr * 0.1}],
12                              lr=self.lr,
13                              momentum=0.9,
14                              weight_decay=1e-4,
15                              nesterov=False)
16 }
```

Строка 1. Скорость обучения.
Строка 2. Кратность увеличения изображения.
Строка 3 — 5. Модель FSRCNN.
Строка 6 — 16. Оптимизатор SGD.

2.2. Обучение

Рисунок 5. Обучение модели

Программа написана на языке Python, где использовалась библиотека Tourch. Программа выполняет обучение модели (рис 5), задано кол-во эпох 100, скорость обучение 0.001. Для выполнения программы использовался графический ускоритель TU117 (GTX 1650).

Для оценки модели применяется метрика MSELoss для обучение, для оценки качество изображения при увеличение разрешение использовалась метрика PSNR для оценки тестов.

Листинг 2.4. Функция для обучение модель

```
1  def step_train(self, sel: any, index :int) -> None:
2          super().step_train(sel, index)
3          model = self.model["fsrcnn"]
4          lr = sel["lr"].to(self.device)
5          hr = sel["hr"].to(self.device)
6          model.zero_grad()
7          with amp.autocast():
8                  sr = model(lr)
9                  loss = self.pixel_criterion(sr, hr)
10         self.gradScaler.scale(loss).backward()
11         self.gradScaler.step(self.optimizer["fsrcnn"])
12         self.gradScaler.update()
13         psnr = 10. * torch.log10(1. / self.psnr_criterion(sr, hr))
14         self.metric.loss = loss.item()
15         self.metric.psnr = psnr.item()
```

Строка 2, заглушка.

Строка 3 — 5, загрузка данных, lr — изображения низкого разрешения, hr — исходный изображения.

Строка 6 — 12. обучаем модель.

Строка 13 — 15. сохраняем метрику в список.

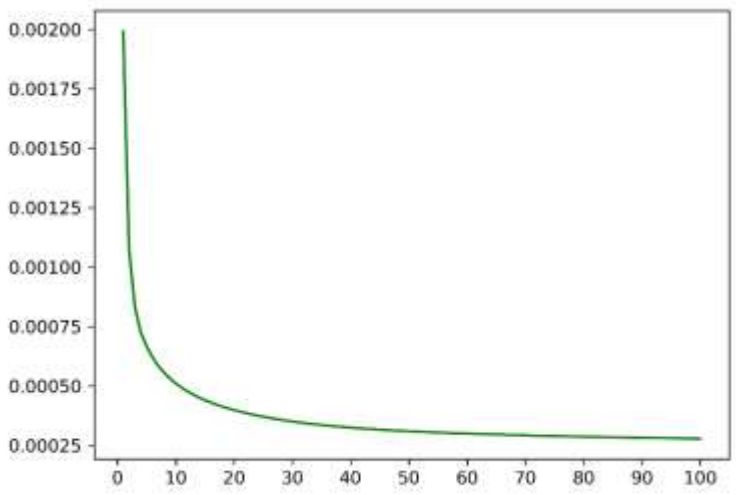

Рисунок 6. Функция потерь (метрика MSELoss)

В результате обучение, получили значение функция потерь (по оси Y) при 100 эпох (по осью X) значение 0.000278 (рис 6). Для оценки качества

улучшения изображения после увеличения размера использовалась функция PSNR, при 100 эпох значение достигла 35.556499 (рис 7).

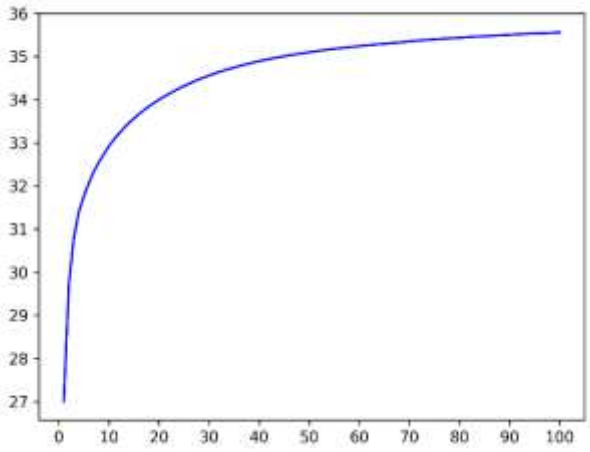

Рисунок 7. Функция PSNR.

$$\mathrm{PSNR}(I, J) = 10 * \log_{10}\left(\frac{MAX_I^2}{\mathrm{MSE}}\right)$$

Рисунок 8. Формула PSNR.

Время выполнение обучение было 5505.294 секунда (1 час, 31 минута, 45.294 секунда).

2.3. Предсказание

Рисунок 9. Слева исходное изображения, справа увеличенное изображение (PSNR 13.7963)

Листинг 2.5. Предобработки изображение для предсказания модели

```
1  self.to_ycbcr = Trans.ColorConvert("rgb", "ycrcb", "cv2")
2  self.from_ycbcr = Trans.ColorConvert("ycrcb", "rgb", "cv2")
3  self.transform_predict = Trans.Compose([
4          self.to_ycbcr,
5          [Trans.Empty(), Trans.Gray(0)],
6          transforms.ToTensor()
7  ])
```

Строка 1 — 2 и 4. Преобразование цвета в YCrCb, и в RGB.

Строка 5. Клонирование изображение, второй изображение в монохромный канал.

Строка 6. Преобразует в тензор.

Листинг 2.6. Функция для предсказания модели, увеличение изображения

```
1  def predict(self, pic :any) -> any:
2          self.model["fsrcnn"].to("cpu").eval()
3          apic = np.array(pic)
4          hr_img, lr_img = self.transform_predict(apic)
5          hr_img = hr_img.unsqueeze(0).to(torch.float32)
6          lr_img = lr_img.unsqueeze(0).to(torch.float32)
7          sr_img = self.model["fsrcnn"](lr_img).clamp_(0.0, 1.0)
8          hr_img = nnf.interpolate(hr_img, size=sr_img.shape[2:], mode='bicubic', align_corners=False)
9          psnr = 10. * torch.log10(1. / torch.mean((sr_img - hr_img[0]) ** 2))
10         hr_img = hr_img.squeeze(0).cpu().detach().numpy() * 255.0
11         sr_img = sr_img.squeeze(0).cpu().detach().numpy() * 255.0
12         res_img = np.moveaxis(np.concatenate((sr_img, hr_img[1:]), axis=0), 0, -1)
13         res_img = self.from_ycbcr(res_img.astype(np.uint8))
14         if type(pic) == Image.Image:
15                 return Image.fromarray(res_img.astype(np.int8), pic.mode), psnr
16         return res_img, psnr
```

Строка 1. Функция выполняет обработку и предсказание модели

Строка 2. Переходит модель на CPU и режим выполнение.

Строка 3 — 6. Обработка изображения и перевод изображение в тензор, подготовка предсказание. Преобразования цвета в YCrCb и извлекает канал Y.

Строка 7. Выполнение модели предсказание обработка изображения. Использует только канал Y.

Строка 8 — 9. Увеличение исходный изображения и оценка метрики PSNR.

Строка 10 — 13. Преобразования от тензора в изображения. Заменяет исходное изображение канала Y, и преобразует в цвета RGB.

Строка 14 — 16. Возвращает результат изображения и метрику.

3 Выводы

В данное статье была использована модель быстрой сверточной нейронной сети сверхвысокого разрешения изображения (FSRCNN) для улучшения качества изображения путем увеличения разрешения, а также использовался набор данных без разметки и выбор случайных изображений разного разрешения для обучения модели. Качество после увеличения изображения использования модели оценивали по метрику PSNR и достигли значения равного 35.556499.

Библиографический список

1. Zhang K., Zuo W., Zhang L. Learning a single convolutional super-resolution network for multiple degradations //Proceedings of the IEEE conference on computer vision and pattern recognition. 2018. С. 3262-3271.
2. Passarella L. S. et al. Reconstructing high resolution ESM data through a novel fast super resolution convolutional neural network (FSRCNN) //Geophysical Research Letters. 2022. Т. 49. №. 4. С. E2021GL097571

3. Park S. et al. Autoencoder-inspired convolutional network-based super-resolution method in MRI //IEEE Journal of Translational Engineering in Health and Medicine. 2021. T. 9. C. 1-13
4. Dong C., Loy C. C., Tang X. Accelerating the super-resolution convolutional neural network //Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14. – Springer International Publishing, 2016. – C. 391-407
5. Chen Y. et al. Brain MRI super resolution using 3D deep densely connected neural networks //2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018). IEEE, 2018. C. 739-742
6. Lee W. et al. Learning with privileged information for efficient image super-resolution //Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIV 16. – Springer International Publishing, 2020. C. 465-482
7. Mane V., Jadhav S., Lal P. Image super-resolution for MRI images using 3D faster super-resolution convolutional neural network architecture //ITM Web of Conferences. – EDP Sciences, 2020. – T. 32. – C. 03044
8. Passarella L. S. et al. Super Resolution Reconstruction of E3SM Data Using a FSRCNN //Authorea Preprints. – 2022
9. Ward C. M. et al. Image quality assessment for determining efficacy and limitations of Super-Resolution Convolutional Neural Network (SRCNN) //Applications of Digital Image Processing XL. SPIE, 2017. T. 10396. C. 19-30
10. Ren H., El-Khamy M., Lee J. CT-SRCNN: cascade trained and trimmed deep convolutional neural networks for image super resolution //2018 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2018. C. 1423-1431
11. Vb S. K. Perceptual image super resolution using deep learning and super resolution convolution neural networks (SRCNN) //Intelligent Systems and Computer Technology. 2020. T. 37. №. 3
12. Aburaed N. et al. 3d expansion of srcnn for spatial enhancement of hyperspectral remote sensing images //2021 4th International Conference on Signal Processing and Information Security (ICSPIS). IEEE, 2021. C. 9-12
13. Yang S., Wang X. Sparse representation and SRCNN based spatio-temporal information fusion method of multi-sensor remote sensing data //Journal of Network Intelligence. 2021. T. 6. №. 1. C. 40-53
14. Prajapati K. et al. Channel split convolutional neural network (ChaSNet) for thermal image super-resolution //Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021. C. 4368-4377
15. Demiray B. Z., Sit M., Demir I. D-SRGAN: DEM super-resolution with generative adversarial networks //SN Computer Science. 2021. T. 2. C. 1-11
16. Zou Y. et al. Super-resolution reconstruction of infrared images based on a convolutional neural network with skip connections //Optics and Lasers in Engineering. 2021. T. 146. C. 106717
17. GitHub - Lornatang/FSRCNN-PyTorch: Fast implementation of fsrcnn algorithm based on pytorch framework // GitHub URL:

https://github.com/Lornatang/FSRCNN-PyTorch (дата обращения: 2023-06-22)

4. Приложения

Листинг 4.1. Исходный код программы

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import torch
import torch.nn as nn
import torch.optim as optim
from torch.cuda import amp
import torch.nn.functional as nnf
from torchsummary import summary
from torchvision import transforms
from typing import Union
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from Lib.AppMain import *
import Lib.Transforms as Trans

#https://github.com/Lornatang/FSRCNN-PyTorch
class FSRCNN(nn.Module):
	def __init__(self, upscale_factor: int) -> None:
		self.upscale_factor = upscale_factor
		super(FSRCNN, self).__init__()
		self.feature_extraction = nn.Sequential(
			nn.Conv2d(in_channels=1, out_channels=56, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2)),
			nn.PReLU(num_parameters=56)
		)
		self.shrink = nn.Sequential(
			nn.Conv2d(in_channels=56, out_channels=12, kernel_size=(1, 1), stride=(1, 1), padding=(0, 0)),
			nn.PReLU(num_parameters=12)
		)
		self.map = nn.Sequential(
			nn.Conv2d(in_channels=12, out_channels=12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
			nn.PReLU(num_parameters=12),
			nn.Conv2d(in_channels=12, out_channels=12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
			nn.PReLU(num_parameters=12),
			nn.Conv2d(in_channels=12, out_channels=12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
			nn.PReLU(num_parameters=12),
			nn.Conv2d(in_channels=12, out_channels=12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
			nn.PReLU(num_parameters=12)
		)
		self.expand = nn.Sequential(
			nn.Conv2d(in_channels=12, out_channels=56, kernel_size=(1, 1), stride=(1, 1), padding=(0, 0)),
			nn.PReLU(num_parameters=56)
		)
		self.deconv = nn.ConvTranspose2d(
			in_channels=56, out_channels=1, kernel_size=(9, 9), stride=(upscale_factor, upscale_factor),
			padding=(4, 4), output_padding=(upscale_factor - 1, upscale_factor - 1))

		self._initialize_weights()

	def forward(self, x: torch.Tensor) -> torch.Tensor:
		out = self.feature_extraction(x)
		out = self.shrink(out)
		out = self.map(out)
		out = self.expand(out)
		out = self.deconv(out)
		return out

	def _initialize_weights(self) -> None:
		for m in self.modules():
			if isinstance(m, nn.Conv2d):
				nn.init.normal_(m.weight.data, mean=0.0, std=np.sqrt(2 / (m.out_channels *
m.weight.data[0][0].numel())))
				nn.init.zeros_(m.bias.data)

		nn.init.normal_(self.deconv.weight.data, mean=0.0, std=0.001)
		nn.init.zeros_(self.deconv.bias.data)
```

```
67
68  class App(AppMain):
69          def main(self):
70                  self.dir_prefix = "./Data"
71                  self.dirs = {
72                          "dataset": "../wallhaven",
73                          "dataset_test": "Test",
74                          "fig": "Fig",
75                  }
76                  self.profile_name = "default"
77                  self.datasets = None
78                  self.model = {}
79                  self.optimizer = {}
80                  self.auto_save_exit = False
81                  self.disp_metric = ["loss", "psnr"]
82                  self.metric.loss = None
83                  self.metric.psnr = None
84
85                  self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
86                  self.psnr_criterion = nn.MSELoss().to(self.device)
87                  self.pixel_criterion = nn.MSELoss().to(self.device)
88                  self.init_dirs()
89
90                  self.lr = 1e-3
91                  self.size_image = 40
92                  self.upscale_factor = 2
93                  self.model = {
94                          "fsrcnn": FSRCNN(self.upscale_factor).to(self.device)
95                  }
96                  self.optimizer = {
97                          "fsrcnn": optim.SGD([{"params": self.model["fsrcnn"].feature_extraction.parameters()},
98                                                                  {"params": self.model["fsrcnn"].shrink.parameters()},
99                                                                  {"params": self.model["fsrcnn"].map.parameters()},
100                                                                 {"params": self.model["fsrcnn"].expand.parameters()},
101                                                                 {"params": self.model["fsrcnn"].deconv.parameters(), "lr":
    self.lr * 0.1}],
102                                                                 lr=self.lr,
103                                                                 momentum=0.9,
104                                                                 weight_decay=1e-4,
105                                                                 nesterov=False)
106                 }
107                 self.resize_image = 80
108                 self.transform = [
109                         Trans.Compose([
110                                 transforms.RandomCrop(self.resize_image),
111                                 transforms.Resize(self.size_image * self.upscale_factor),
112                                 transforms.RandomRotation(360),
113                                 Trans.ColorConvert("rgb", "ycrcb", "cv2"),
114                                 transforms.Grayscale(num_output_channels=1),
115                                 [Trans.Empty(), transforms.Resize(self.size_image)],
116                                 transforms.ToTensor()
117                         ])
118                 ]
119                 self.to_ycbcr = Trans.ColorConvert("rgb", "ycrcb", "cv2")
120                 self.from_ycbcr = Trans.ColorConvert("ycrcb", "rgb", "cv2")
121                 self.transform_predict = Trans.Compose([
122                         self.to_ycbcr,
123                         [Trans.Empty(), Trans.Gray(0)],
124                         transforms.ToTensor()
125                 ])
126                 self.gradScaler = amp.GradScaler()
127                 self.add_datasets(transform=self.transform, count_files=512, batch_size=16, count=100, header=["hr", "lr"])
128         def start_train(self) -> None:
129                 self.model["fsrcnn"].train()
130         def step_train(self, sel: any, index :int) -> None:
131                 super().step_train(sel, index)
132                 model = self.model["fsrcnn"]
133
134                 lr = sel["lr"].to(self.device)
135                 hr = sel["hr"].to(self.device)
136
137                 model.zero_grad()
138                 with amp.autocast():
139                         sr = model(lr)
140                         loss = self.pixel_criterion(sr, hr)
141
142                 self.gradScaler.scale(loss).backward()
143                 self.gradScaler.step(self.optimizer["fsrcnn"])
```

```
144                         self.gradScaler.update()
145
146                         psnr = 10. * torch.log10(1. / self.psnr_criterion(sr, hr))
147                         self.metric.loss = loss.item()
148                         self.metric.psnr = psnr.item()
149
150             def gen_image_color(self):
151                         img = np.full([64, 64, 3], np.random.randint(0, 255, 3)).astype('uint8')
152                         return Image.fromarray(img, 'RGB')
153             def load_image(self, path :str) -> Image.Image:
154                         with open(path, "rb") as f:
155                                     img = Image.open(f)
156                                     return img.convert("RGB")
157             def save_image(self, path :str, img :Union[np.ndarray, Image.Image]) -> None:
158                         pic = None
159                         with open(path, "wb") as f:
160                                     if type(img) != Image.Image:
161                                                 pic = Image.fromarray(img, "RGB")
162                                     else:
163                                                 pic = img
164                                     pic.save(f)
165             def predict(self, pic :any) -> any:
166                         self.model["fsrcnn"].to("cpu").eval()
167                         apic = np.array(pic)
168                         hr_img, lr_img = self.transform_predict(apic)
169                         hr_img = hr_img.unsqueeze(0).to(torch.float32)
170                         lr_img = lr_img.unsqueeze(0).to(torch.float32)
171
172                         sr_img = self.model["fsrcnn"](lr_img).clamp_(0.0, 1.0)
173                         hr_img = nnf.interpolate(hr_img, size=sr_img.shape[2:], mode='bicubic', align_corners=False)
174                         psnr = 10. * torch.log10(1. / torch.mean((sr_img - hr_img[0]) ** 2))
175                         hr_img = hr_img.squeeze(0).cpu().detach().numpy() * 255.0
176                         sr_img = sr_img.squeeze(0).cpu().detach().numpy() * 255.0
177                         res_img = np.moveaxis(np.concatenate((sr_img, hr_img[1:]), axis=0), 0, -1)
178                         res_img = self.from_ycbcr(res_img.astype(np.uint8))
179                         if type(pic) == Image.Image:
180                                     return Image.fromarray(res_img.astype(np.int8), pic.mode), psnr
181                         return res_img, psnr
182             def predict_test(self, pic :any) -> any:
183                         self.model["fsrcnn"].to("cpu").eval()
184                         apic = np.array(pic)
185                         hr_img, lr_img = self.transform_predict(apic)
186                         hr_img = hr_img.unsqueeze(0).to(torch.float32)
187                         lr_img = lr_img.unsqueeze(0).to(torch.float32)
188
189                         lr_img = nnf.interpolate(lr_img, size=tuple((np.array(lr_img.shape[2:]) /
    self.model["fsrcnn"].upscale_factor).astype("int32")), mode='bicubic', align_corners=False)
190                         sr_img = self.model["fsrcnn"](lr_img).clamp_(0.0, 1.0)
191                         psnr = 10. * torch.log10(1. / torch.mean((sr_img - hr_img[0]) ** 2))
192                         hr_img = hr_img.squeeze(0).cpu().detach().numpy() * 255.0
193                         sr_img = sr_img.squeeze(0).cpu().detach().numpy() * 255.0
194                         res_img = np.moveaxis(np.concatenate((sr_img, hr_img[1:]), axis=0), 0, -1)
195                         res_img = self.from_ycbcr(res_img.astype(np.uint8))
196                         if type(pic) == Image.Image:
197                                     return Image.fromarray(res_img.astype(np.int8), pic.mode), psnr
198                         return res_img, psnr
199
200 def predict_sp(image, size = (40, 40), padding=[0, 0, 0, 0]):
201             grid = Trans.Grid(size)
202             grid_full = Trans.Merge(padding=padding)
203             m = grid(image)
204             d_metric = deque()
205             for i in range(len(m)):
206                         r = app.predict(m[i])
207                         d_metric.append(float(r[1]))
208                         m[i] = r[0]
209             img_m = grid_full(m)
210             return img_m, d_metric
211
212 def predict_sp_test(image, size = (40, 40), padding=[0, 0, 0, 0]):
213             grid = Trans.Grid(size)
214             grid_full = Trans.Merge(padding=padding)
215             m = grid(image)
216             d_metric = deque()
217             for i in range(len(m)):
218                         r = app.predict_test(m[i])
219                         d_metric.append(float(r[1]))
220                         m[i] = r[0]
```

```
                img_m = grid_full(m)
                return img_m, d_metric


app = App()
app.main()
app.load_datasets(cache=True)
app.save_datasets("Fsrcnn")

# Train
summary(app.model["fsrcnn"], input_size=(1, 40, 40))
app.fit(100)
app.save("epoch100")

# Predict
app.load_model("epoch100")

sel = app.load_image("./lena.png")
c = predict_sp_test(sel, size=(100, 100), padding=[1, 1, 1, 1])
print(np.average(c[1]))
c[0].save("./sel_predict_test.jpeg")

c = app.predict_test(sel)
print(float(c[1]))
c[0].save("./sel_app_predict_test.jpeg")

# Plotting
df = pd.DataFrame(app.metric._data)
x_step = 10

fig, ax = plt.subplots()
ax.plot(df["Epoch"], df["Time"], label ='Time', color="red")
ax.set_xticks(np.arange(0, len(df)+1, x_step))
fig.savefig(app.get_path("fig", "{0}.png".format("Time")), dpi = 300)

fig, ax = plt.subplots()
ax.plot(df["Epoch"], df["loss"], label ='loss', color="green")
ax.set_xticks(np.arange(0, len(df)+1, x_step))
fig.savefig(app.get_path("fig", "{0}.png".format("loss")), dpi = 300)

fig, ax = plt.subplots()
ax.plot(df["Epoch"], df["psnr"], label ='psnr', color="blue")
ax.set_xticks(np.arange(0, len(df)+1, x_step))
ax.set_yticks(np.arange(np.floor(min(df["psnr"])), np.ceil(max(df["psnr"]))+1, 1))
fig.savefig(app.get_path("fig", "{0}.png".format("psnr")), dpi = 300)
```