neural image codec

Прошян Гарри Арменович M4150

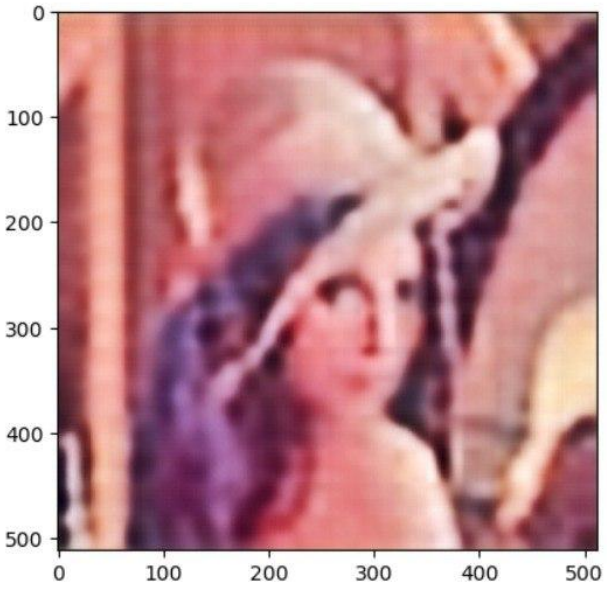
Модель

- Энкодер - все сверточные слои resnet18
- Декодер - “зеркальная версия” энкодера, состоящая из последовательности residual блоков вида:

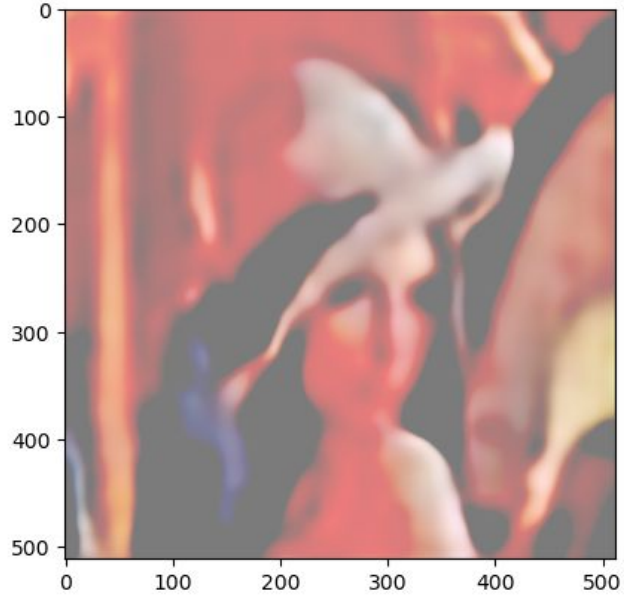
```
def forward(self, x):
        skip_connection = self.conv_to_match_dims(x)
        out = self.upscale(x)
        out = self.double_conv(out)
        out = out + skip_connection
        return out
```

  - double_conv = (Conv2d -> Batchnorm -> Relu) x2
  - upscale = upsample
  - conv_to_match_dims - transpose convolution

Прирост качества при residual декодере*



residual decoder



sequential decoder

* Оба автоэнкодера с данного слайда обучались 5 эпох

Квантование (только инференс)

Энкодер возвращает значения от 0 до 1 (ФА сигмоида)

Данные значения квантуются в целые числа из интервала [0 до 2^B]

```
def quantize(encoder_out: torch.Tensor, B: int):
    quantized = torch.round(encoder_out * 2**B)
    return quantized.type(torch.int8)
```

Функция потерь - MSE

Класс NeuralImageCompressor

```python
class NeuralImageCompressor(nn.Module):
    def __init__(self,
                 encoder: Encoder,
                 decoder: nn.Module,
                 B: int = 1):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.B = B

    def _get_quantization_error(self, shape: Tuple[int, ...]):
        mean = torch.full(shape, -0.5)
        std = torch.full(shape, 0.5)
        quan_err = 0.5**self.B * torch.normal(mean = mean, std = std)
        return quan_err

    def forward(self, x):
        out = self.encoder(x)
        quant_err = self._get_quantization_error(out.shape).to(out.device)
        out = out + quant_err
        out = self.decoder(out)
        return out
```

При обучении модели к выходу энкодера прибавлялся шум, соизмеримый по амплитуде с ошибкой квантования

LooselessCompressor

Для энтропийного кодирования используется объект наследника абстрактного класса LooselessCompressor. Данный абстрактный класс имеет 5 чисто виртуальных методов:

- инициализация состояния алгоритма сжатия из файла
- инициализация состояния алгоритма сжатия по последовательности
- кодирование
- декодирование
- сохранение состояния в файл

Huffman

Реализует алгоритм Хаффмана, является наследником LooselessCompressor.

При использовании в encoder_pipeline сохраняет свое состояние в **отдельный** .json файл. JSON был выбран для человеко-читаемости, ясно, что закодировать дерево в бинарном виде было бы экономнее

При использовании в decoder_pipeline состояние считывается из переданного файла

В файл с сжатым изображением не входит информация о состоянии алгоритма энтропийного кодирования

Пример .json с состоянием алгоритма Хаффмана

```
{"16": "00000", "15": "00001", "14": "0001", "13": "001", "8": "01",
"12": "100", "9": "101", "10": "110", "11": "111"}
```

Это .json для peppers при B = 4. Можно заметить, что значения ниже 8 не присутствуют

Было проверено, что энкодер ни для одного изображения из [peppers.png, lena.png, baboon.png] не возвращает значения ниже 0.4

Я прибавляю шум и на инференсе

При малых B если не прибавлять после квантизации и “деквантизации” (умножения на 2^B) шум, изображения получаются искаженными.

При этом искажения аналогичны тем, что будут, если не делать квантизацию и не прибавлять шум

Датасет

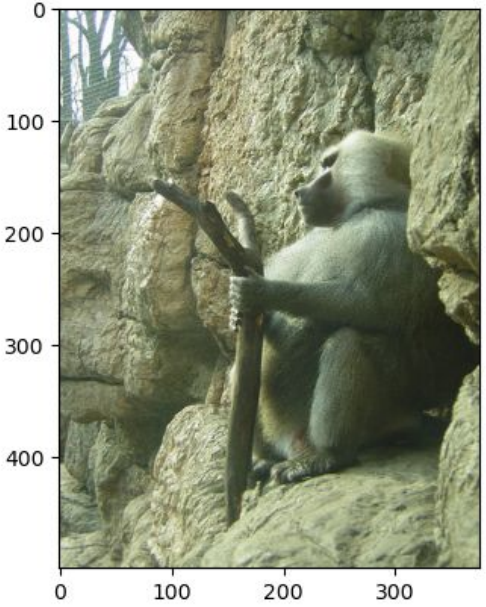
17’000 изображений из 13 классов из Imagenet:

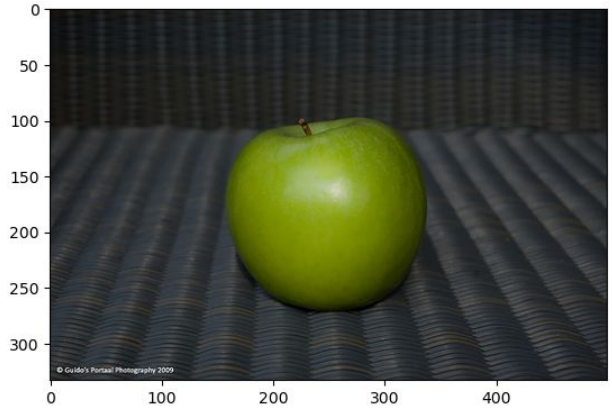
```
"n02486410": "baboon",
"n07720875": "bell pepper",
"n03124170": "cowboy hat, ten-gallon hat",

"n02493509": "titi, titi monkey",
"n02110627": "affenpinscher, monkey pinscher, monkey dog",
"n02493793": "spider monkey, Ateles geoffroyi",
"n02480855": "gorilla, Gorilla gorilla",

"n07742313": "Granny Smith",
"n03724870": "mask",
"n03379051": "football helmet",
"n04356056": "sunglasses, dark glasses, shades",
"n04591157": "Windsor tie",
"n02906734": "broom"
```

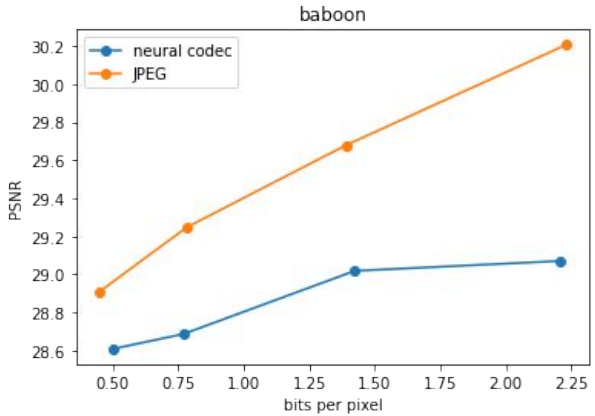
Датасет

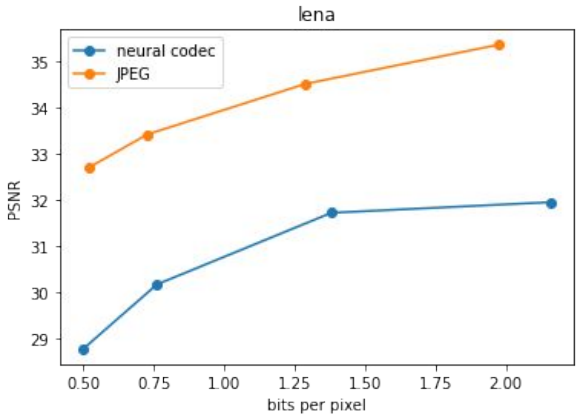

0
100
200
300
400
0
100
200
300

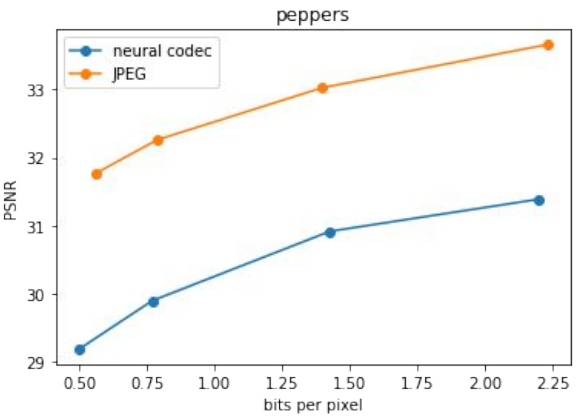

0
50
100
150
200
250
300
0
100
200
300
400


0
50
100
150
200
250
300
350
400
0
100
200
300

Результаты


baboon
neural codec
JPEG
PSNR
bits per pixel
30.2
30.0
29.8
29.6
29.4
29.2
29.0
28.8
28.6
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25


lena
neural codec
JPEG
PSNR
bits per pixel
35
34
33
32
31
30
29
0.50
0.75
1.00
1.25
1.50
1.75
2.00


peppers
neural codec
JPEG
PSNR
bits per pixel
33
32
31
30
29
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25

B = 6 сравнение с JPEG

jpg

q = 70, bpp = 2.027, psnr = 30.080

jpg

q = 90, bpp = 1.975, psnr = 35.357

jpg

q = 90, bpp = 2.229, psnr = 33.656

neural codec

B = 6, bpp = 2.204, psnr = 29.209

neural codec

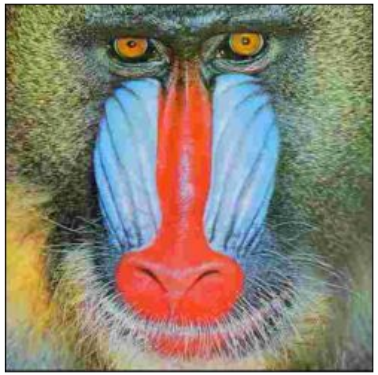
B = 6, bpp = 2.158, psnr = 32.401

neural codec

B = 6, bpp = 2.195, psnr = 31.867

B = 1 сравнение с JPEG

jpg



q = 10, bpp = 0.448, psnr = 28.907

jpg

q = 30, bpp = 0.521, psnr = 32.706

jpg

q = 30, bpp = 0.562, psnr = 31.768

neural codec

B = 1, bpp = 0.500, psnr = 28.627

neural codec

B = 1, bpp = 0.500, psnr = 28.745

neural codec

B = 1, bpp = 0.500, psnr = 29.230