

# 1 Самообучение: раскраска изображений (image colorization)

Выполнил Оганов Александр Артурович

## 1.1 Используемые статьи

Перечислим статьи и кратко опишем статьи, которые использовались в работе 1. [Colorful Image Colorization](#) - постановка задачи раскраски изображений и сравнение задачи классификации цвета с задачей регрессии (предсказание цвета). Автор рассматривает задачу предсказания каналов  $a$ ,  $b$  по каналу  $L$  [Lab color space](#), то есть по черно-белому изображению построить цветное;

2. [Split-Brain Autoencoders: Unsupervised Learning by Cross-Channel Prediction](#) - обобщение задачи раскраски изображений на произвольные каналы, в качестве основной модели был взят автокодировщик;
3. [Analysis of Different Losses for Deep Learning Image Colorization](#) - исследование влияния функции потерь для задачи раскраски изображений, перечеслены основные подходы к решению (классификация, регрессия, приближение распределений, использование GAN) задачи;
4. [The Unreasonable Effectiveness of Deep Features as a Perceptual Metric](#) - построение метрики (на основе нейросети), которая отражает похожесть изображений для человека, полученную метрику автор статьи называет lpips.

## 2 Подход к решению поставленной задачи

Решать задачу будем на датасете Food101, выбор обоснован красочностью датасета и его размерами.

В качестве основной модели, следуя статье [Split-Brain Autoencoders: Unsupervised Learning by Cross-Channel Prediction](#), будем использовать автокодировщик, а в качестве эмбедингов - его латентное пространство. Код для обучения модели представлен в ноутбуке “my colorization.ipynb”, для удобства были использованы блоки из библиотеки diffusers, а именно [Encoder](#) и [Decoder](#). Архитектура модели описана в файле “Autoencoder.py”.

В качестве функции потерь была взята комбинация двух функций. В след за статьей [Colorful Image Colorization](#) мы использовали [smooth L1 loss](#) для попиксельного сравнения и lpips для схожести изображений.

Для оценки полученных представлений проведем следующий эксперимент на датасете Food101. Мы разделим обучающую выборку в соотношение 9:1 для обучения автокодировщика и классификатора соответственно. Также из обучения автокодировщика исключим 20 классов из 101. Тогда наш эксперимент будет выглядеть следующим образом:

Обучение автокодировщика на 81 классе. Обучение классификатора на 1/10 обучающей выборки в латентном пространстве автокодировщика и его сравнение с классификатором, который обучался на 1/10 обучающей выборки

Тем самым мы получим ответы на следующие вопросы:

- Помогают ли выученные представления в достижении лучшего результата классификации при наличии маленькой обучающей выборки (Эксперимент 1)?
- Насколько хорошо обобщает данные автокодировщик и влияет ли на результаты классы, которые использовались в его обучении?

Так как нашей целью не является получение высокого качества классификации, мы будем использовать простой сверточный классификатор ResNetFoodClassifierSmall:

```

class BasicBlock(nn.Module):
    def __init__(self, in_channels, hidden, out_channels):
        super().__init__()
        self.basic_block = nn.Sequential(
            nn.Conv2d(in_channels, hidden, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(hidden, eps=1e-05),
            nn.ReLU(inplace=True),
            nn.Conv2d(hidden, out_channels, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels, eps=1e-05)
        )

    def forward(self, x):
        return self.basic_block(x)

class ResNetFoodClassifierSmall(nn.Module):
    def __init__(self, prefix, in_channels, classes):
        super().__init__()

        self.back_bone_prefix = prefix

        for parametr in self.back_bone_prefix.parameters():
            parametr.requires_grad = False

        self.basic_block = BasicBlock(in_channels, 128, 256)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.out = nn.Linear(256, classes)

    def forward(self, x):
        x = self.back_bone_prefix(x)
        x = self.basic_block(x)
        x = self.avg_pool(x)
        x = x.view(x.size(0), x.size(1))
        x = self.out(x)

        return x

```

### 3 Датасет

Датасет Foof101 содержит 101 класс разной еды, каждый класс содержит 750 экземпляров в обучающей выборке и 250 в валидационной. [Подробнее о датасете](#)



### 4 Обучение автокодировщика

Параметры обучения:

```
w1 = 1  
w2 = 1
```

```
pixel_loss = nn.SmoothL1Loss()  
perc_loss = lpip.LearnedPerceptualImagePatchSimilarity()
```

```
def loss_fn(x, y):
```

```

l1 = w1 * pixel_loss(x, y)
l2 = w2 * perc_loss(x, y)
return l1 + l2, l1.item() / w1, l2.item() / w2

model = Autoencoder()
optimizer = optim.AdamW(model.parameters(), lr=0.0001, weight_decay=2e-05)

```

Число параметров модели: 1287631, обучение проходило 16 эпох на 9/10 обучающей выборки за исключением 20 классов.

В качестве исключенных классов были выбраны: ‘guacamole’, ‘spring\_rolls’, ‘carrot\_cake’, ‘paella’, ‘lobster\_bisque’, ‘chicken\_wings’, ‘ravioli’, ‘sashimi’, ‘peking\_duck’, ‘peking\_duck’, ‘scallops’, ‘tuna\_tartare’, ‘churros’, ‘baklava’, ‘chocolate\_cake’, ‘gyoza’, ‘baby\_back\_ribs’, ‘scallops’, ‘cup\_cakes’, ‘filet\_mignon’

Подробнее код продемонстрирован в ноутбуке “my\_colorization.ipynb”, история обучения и графики находятся по [ссылке](#).

Ниже представлены примеры на обучающем датасете, на валидации и на 20 классах исключенных из обучения соответственно



Общее качество раскрашенных изображений совпадает с исходной выборкой, но по цвету преобладают желтые тона. Так как нашей целью являлось получение хорошо обученных представлений, мы не будем менять архитектуру и/или усложнять обучение.

## 5 Обучение классификаторов

Будем обучать классификаторы отдельно на классах, которые использовались в обучение автокодировщика (на графиках in-domain), и на остальных (на графиках out-of-domain). В качестве классификатора будем использовать ResNetFoodClassifierSmall, для обучения на исходной выборке:

```

model = ResNetFoodClassifierSmall(nn.Identity(), 3, 101)

optimizer = optim.AdamW(model.parameters(), lr=0.0003, weight_decay=2e-05)

Для обучения на латентном пространстве (выученных представлениях):

```

```
model = ResNetFoodClassifierSmall(autoencoder.encoder, 4, 101)

optimizer = optim.AdamW(model.parameters(), lr=0.0003, weight_decay=2e-05)
```

Число параметров модели: 325093, обучение проходило 10 эпох на 1/10 обучающей выборки. В качестве функции потерь будем использовать кросс-энтропию.

Код обучения находится в ноутбуке “classifier small.ipynb”, история обучения [для in-domain](#) и [для out-of-domain](#)

На графиках видно, что классификатор обучаемый на латентном пространстве достигает меньших значений лосса в обоих случаях. На основании этого мы можем сделать вывод, что наш подход действительно повышает качество модели, то есть решает задачу самообучения, а именно автокодировщик выучивает нужные для классификации представления.

## 6 Выводы

На наш взгляд, основными выводами, которые можно извлечь из работы, являются:

1. Использование lpips метрики и smooth L1 loss позволяет хорошо решать задачу раскраски изображений;
2. Описанный выше эксперимент продемонстрировал, что выученные представления повышают качество как на обучении так и на валидации;
3. Автокодировщик обладает хорошей обобщающей способностью и выученные представления повышают качество на классах, которые не использовались в обучении.

## 7 Идеи

В рамках работы не удалось проверить некоторые идеи, которые не относятся к задаче самообучения на прямую, а именно:

1. Влияние функции потерь для классификации цвета из статьи [Colorful Image Colorization](#);
2. Использование в качестве выученных представлений не латентное пространство, а активации других слоев автокодировщика;
3. Добавление ядерной регуляризации для уменьшения переобучения рассмотренных моделей;
4. Использование более сложных архитектур для классификации с целью получения максимального качества на маленькой обучающей выборке.