

Дипломная работа

На тему: **“Style Transfer with GANs”**

Майстренко Александра

В рамках курса “Machine Learning”

Старт обучения: 23.05.2020

Преподаватель: Филипп Кофман

Школа: ithillel.ua

Выбор моделей (обзор)

1. CycleGAN

Статья: <https://junyanz.github.io/CycleGAN/>

2. MUNIT

<https://github.com/NVlabs/MUNIT>

3. UNIT

<https://github.com/NVlabs/imaginaire/blob/master/projects/unit/README.md>

4. FUNIT

<https://github.com/NVlabs/FUNIT> (пример - изменение породы собаки)

Обзорная статья

[Awesome transfer learning \(GitHub\)](#)

Статья содержит список алгоритмов **Image-to-Image translation**, а также ссылки на [датасеты](#)

- DIAT: [Deep Identity-aware Transfer of Facial Attributes](#) (2016)
- Pix2pix: [Image-to-Image Translation with Conditional Adversarial Networks](#) (2016)
- DTN: [Unsupervised Cross-domain Image Generation](#) (2016)
- SimGAN: [Learning from Simulated and Unsupervised Images through Adversarial Training \(2016\)](#) (2016)
- PixelDA: [Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks](#) (2016)
- UNIT: [Unsupervised Image-to-Image Translation Networks](#) (2017)
- CycleGAN: [Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks](#) (2017)
- DiscoGAN: [Learning to Discover Cross-Domain Relations with Generative Adversarial Networks](#) (2017)
- DualGAN: [DualGAN: Unsupervised Dual Learning for Image-to-Image Translation](#) (2017)
- SBADA-GAN: [From source to target and back: symmetric bi-directional adaptive GAN](#) (2017)
- DistanceGAN: [One-Sided Unsupervised Domain Mapping](#) (2017)
- pix2pixHD: [High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs](#) (2018)
- I2I: [Image to Image Translation for Domain Adaptation](#) (2017)
- MUNIT: [Multimodal Unsupervised Image-to-Image Translation](#) (2018)
- LSTNet: [Unsupervised Latent Space Translation Network](#)(2020)

CycleGAN

Алгоритм в основе - <http://shikib.com/CycleGan.html>

Discriminator

Input shape torch.Size([2, 3, 126, 126])

ORIGINAL CODE: (70 x 70)

Conv2d output shape: torch.Size([2, 32, 62, 62])

Conv2d output shape: torch.Size([2, 64, 30, 30])

Conv2d output shape: torch.Size([2, 128, 14, 14])

BatchNorm2d output shape: torch.Size([2, 128, 14, 14])

Conv2d output shape: torch.Size([2, 256, 6, 6])

BatchNorm2d output shape: torch.Size([2, 256, 6, 6])

Conv2d output shape: torch.Size([2, 512, 2, 2])

BatchNorm2d output shape: torch.Size([2, 512, 2, 2])

Conv2d output shape: torch.Size([2, 512, 1, 1])

Linear output shape: torch.Size([2, 1]) # nn.Linear(512, 1)

Output shape torch.Size([2, 1])

Discriminator layers

Описание слоев

```
discriminator_126(  
    (conv0): Conv2d(3, 32, kernel_size=(4, 4), stride=(2, 2))  
    (conv1): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))  
    (conv2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2))  
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (conv3): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2))  
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (conv4): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2))  
    (bn4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (conv5): Conv2d(512, 512, kernel_size=(2, 2), stride=(1, 1))  
    (head): Linear(in_features=512, out_features=1, bias=True)  
)
```

Кол-во обучаемых параметров

```
sum(p.numel() for p in d_out.parameters() if p.requires_grad)  
3839201
```

Generator 1

image_size = (272, 480), 130,560 pix

Кол-во обучаемых параметров **1,968,777**

```
generator1(  
    (r1): ReflectionPad2d((3, 3, 3, 3))  
    (conv1): Conv2d(3, 32, kernel_size=(7, 7), stride=(1, 1)) +BatchNorm2d  
    (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1)) +BatchNorm2d  
    (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1)) +BatchNorm2d
```

12 x RESIDUAL blocks

```
    (r4): ReflectionPad2d((1, 1, 1, 1))  
    (conv4): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))  
    (bn4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    X = X + X1  
  
    (uconv16): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2)) +BatchNorm2d  
    (uconv17): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(2, 2)) +BatchNorm2d  
    (r18): ReflectionPad2d((3, 3, 3, 3))  
    (conv18): Conv2d(32, 3, kernel_size=(7, 7), stride=(1, 1)) +BatchNorm2d  
)
```

Generator 2

image_size = (272, 480), **130,560 pix**

Кол-во обучаемых параметров **26,433**

```
Generator(                                                                 #BatchNorm2d and ReLU are hidden
  (0): ReflectionPad2d((3, 3, 3, 3))
  (0): Conv2d(3, 3, kernel_size=(7, 7), stride=(1, 1), bias=False)
  (0): Conv2d(3, 6, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (0): Conv2d(6, 12, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
```

9 X ResiduleBlock

```
  (0): ReflectionPad2d((1, 1, 1, 1))
  (0): Conv2d(12, 12, kernel_size=(3, 3), stride=(1, 1), bias=False)
  (2): ReflectionPad2d((1, 1, 1, 1))
  (3): Conv2d(12, 12, kernel_size=(3, 3), stride=(1, 1))

  (0): ConvTranspose2d(12, 6, kernel_size=(3, 3), stride=(2, 2))
  (0): ConvTranspose2d(6, 3, kernel_size=(3, 3), stride=(2, 2))
(15): ReflectionPad2d((3, 3, 3, 3))
(16): Conv2d(3, 3, kernel_size=(7, 7), stride=(1, 1))
(17): Tanh()
)
```


Моя LOSS функция для генератора

Постановка задачи для
процесса обучения через Loss
функцию :)

Слушайся дискриминатора

```
def generator_loss(scores_fake, label_real, photo_restruct, photo_real):  
    return 5 * torch.mean((scores_fake - label_real)**2) \  
        + 7 * torch.mean(torch.abs(photo_restruct - photo_real)) \  
        + 15 * torch.mean(torch.abs(photo_restruct - photo_real)**2)
```

Верните со вторым генератором
изначальную картинку (точно)

Сильное наказание, если картинка
совсем не похожа

Самое интересное - проблемы

Поиск и подготовка данных, их объем

Ошибки в коде

Размер картинки

Ограниченные ресурсы

Подготовка данных

Фото с регистраторов:

110 тыс картинок

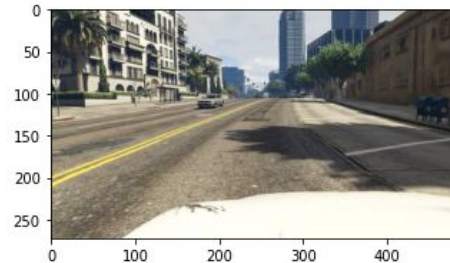
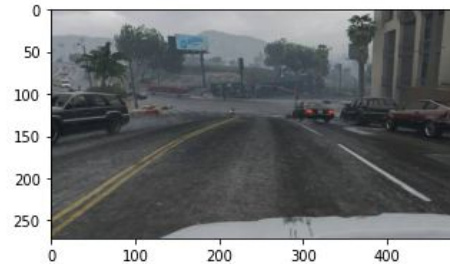
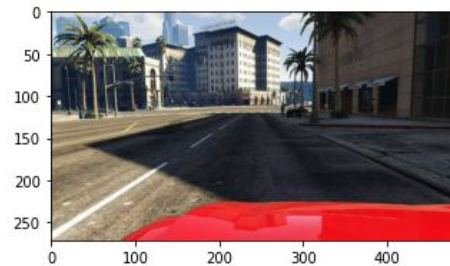
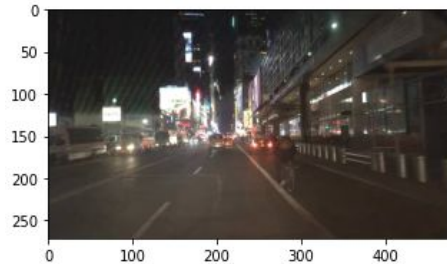
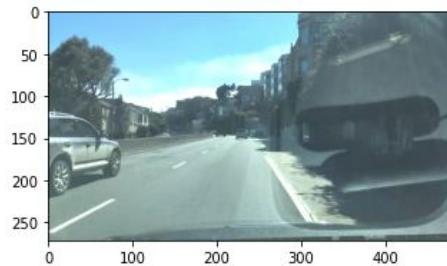
Особенности:

- Много ночных
- Есть зима

GTA 5:

25 тыс картинок

- Коричневый асфальт
- Желтая разметка
- Сиреневое небо
- Чаше красный светофор



Объем данных

Google disk: 15 Gb

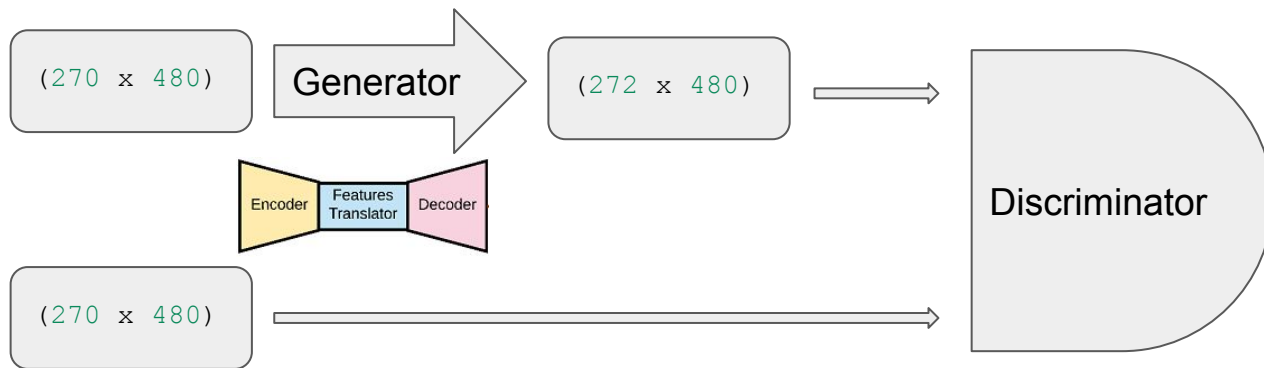
Домашний диск (свободно): 40 Gb

Объем необработанных фото: 60 + 10 Gb (10 архивов + 1)

Решение: Привести к размеру **270 x 480** частями на гугл диске
(конвертер zip -> zip)

Результирующий размер базы: 3,6 Gb

Размер картинки при подготовке данных



RuntimeError: The size of tensor a (272) must match the size of tensor b (270) at non-singleton dimension 2

$270/2 = 135$ (нечетное)

Добрый индус

Лучше всего алгоритм
усваивается при
исправлении ошибок в
коде

```
#Train Dg
photo_fake = G(photo_real)

scores_real = pass_through_discriminator(Dg, photo_real)
scores_real_np = Dgnp(photo_real)
scores_fake = pass_through_discriminator(Dg, photo_fake)
scores_fake_np = Dgnp(photo_fake)

label_fake = Variable(torch.zeros(batch_size)).type(dtype)
label_real = Variable(torch.ones(batch_size)).type(dtype)

scores_real = (0.8 * scores_real + 0.2 * scores_real_np)
scores_fake = (0.8 * scores_fake + 0.2 * scores_fake_np)

loss1 = torch.mean((scores_real - label_real)**2)
loss2 = torch.mean((scores_fake - label_fake)**2)

Dg_optim.zero_grad()

loss_dg = (loss1 + loss2)
if batch % 100 == 0:
    print 'Discriminator G loss: {}'.format(loss_dg.data[0])
loss_dg.backward()

Dg_optim.step()

#Train G
photo_fake = G(photo_real)

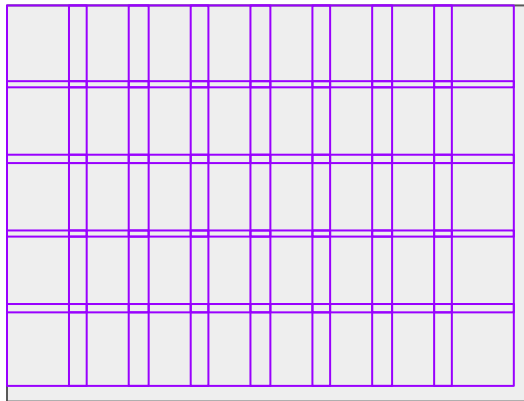
scores_fake = pass_through_discriminator(Dg, photo_fake)
loss_g = torch.mean((scores_fake - label_real)**2) + 10 * torch.mean(torch.abs(G(F(photo_real)) - photo_real))
if batch % 100 == 0:
    print 'Generator G loss: {}'.format(loss_g.data[0])

G_optim.zero_grad()
loss_g.backward()
G_optim.step()
```

Проблема прямоугольной картинки

Картинка: 272 x 480

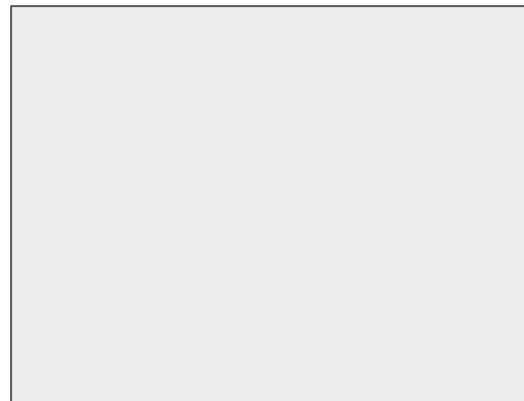
Дискриминатор: 64 x 64



Дискриминатор



Картинка



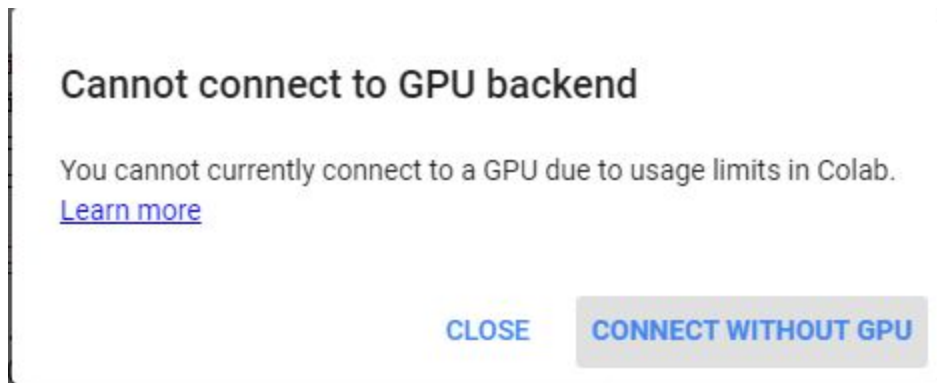
Обход картинки дискриминатором

```
def pass_through_discriminator(discriminator, image):  
    score, k = 0, Variable(torch.zeros(1)).type(dtype)  
    xp, yp = 0, 0  
    x, y = 70, 70  
    offset = 25  
  
    while x < 128:  
        while y < 128:  
            k += 1  
            score += discriminator(image[:, :, xp:x, yp:y])  
            yp += offset  
            y += offset  
  
            xp += offset  
            x += offset  
  
    return score / k
```

Найди 2 ошибки :)

Мало ресурсов

17 эпох * 4000 картинок за 8 часов -> за 8 часов не пробегает весь датасет



Размер сетей + размер картинки


Размер картинки: $272 \times 480 = 130$ тыс

Кол-во обучаемых параметров дискриминатора 2,509,623

Кол-во обучаемых параметров генератора 2,765,568

Batch size = 8 :(

Active sessions

Title		Last execution	RAM used	GPU used	
	Diplom_CycleGAN-2.ipynb Current session	GPU	4 minutes ago	3.35 GB	10.50 GB TERMINATE

Скорость обучения

Для оценки качества написанного решения нужно от нескольких часов



До 3-х раундов по 8 часов обучения:



Следим за размерностью (assert is welcome)

```
loss1 = torch.mean((scores_real - label_real)**2)
```

```
scores_real = tensor([[0.4820],  
    [0.4318],  
    [0.4423],  
    [0.5089],  
    [0.5253],  
    [0.4829],  
    [0.4256],  
    [0.5068]], device='cuda:0', grad_fn=<DivBackward0>)  
label_real = tensor([1., 1., 1., 1., 1., 1., 1., 1.], device='cuda:0')
```

Борьба с яркостью

Оригинал (Game) ->
ярко



0_1500200_photo2_cp...

Fake (Real)
темно



0_1500200_photo2_f.j...

->

Game (Restore)
ярко



0_1500200_photo2_re...

Применил ко всем картинкам из реального мира:

```
if self.adj_brightness:
    image = torchvision.transforms.functional.adjust_contrast(image, 0.7)
    image = torchvision.transforms.functional.adjust_brightness(image, brightness_factor = 1.5)
```

Победа над яркостью

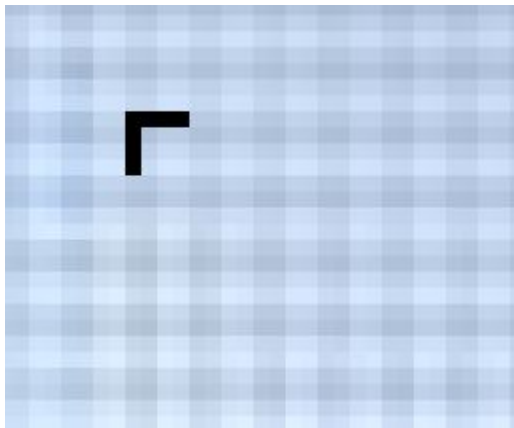
достаточно было обучения на 2500 пар картинок



Баланс между сохранением стиля и наполнения



Небо с шахматными артефактами



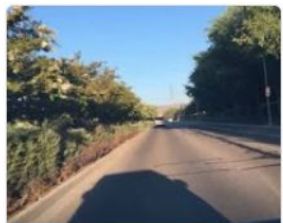
This approach minimize checkerboard artifacts during training:

Use nearest-neighbor resized convolutions instead of strided convolutions

- <https://distill.pub/2016/deconv-checkerboard/>
- <https://github.com/abhiskk/fast-neural-style>

Fails

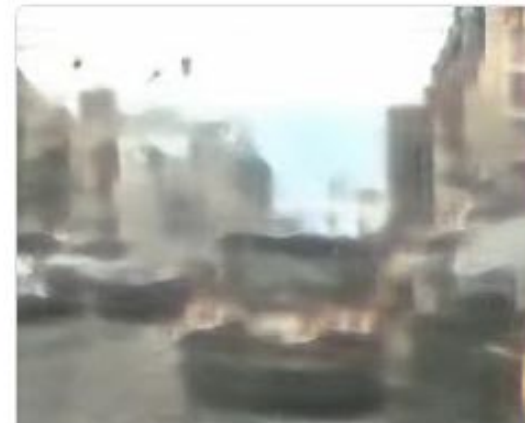
Негативные фото (большая сеть)



Большая модель так и не обучилась (плохой дискриминатор)

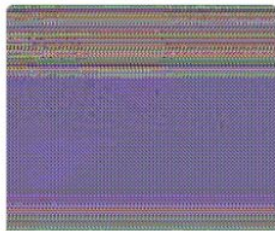


Fails-2. Проблемы в Loss функции



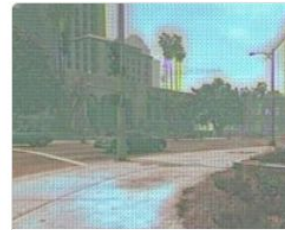
Fail-3

Сеть зашифровала? При этом восстанавливается начальная картинка



Achievements

Первая зеленая трава



Achievements-2

Серый асфальт, сиреневое небо



Небо - синее, деревья - зелёные



Achievements-3

Идеальное восстановление при длительном обучении



Сеть справилась с желтой дорожной разметкой



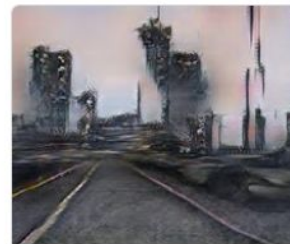
Лучшая сеть



Интересные преобразования

День -> Ночь (и наоборот)

В игре мало ночных кадров



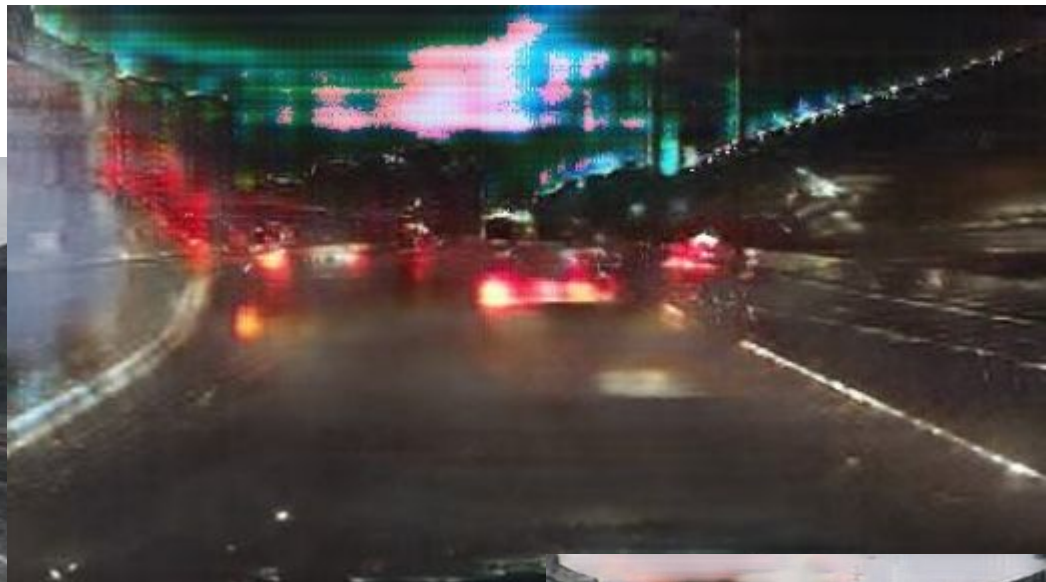
Опасные моменты

Смена цвета светофора :)



Еще больше Generator - художник

Разметка, ночь, дорисованы машины



Log (напоминает фильм Матрица)

Epoch number: 4

****Batch number: 100****

Discriminator G loss: **0.6020406559109688**

Generator G loss (from discr + from generat): **0.11320358894765377 + 0.08016358196735382**

Generator G loss (from generator items example): [0.04839184507727623, 0.04217105358839035]

Discriminator F loss: 0.641342505812645

Generator F loss (from discr + from generat): 0.09631149031221867 + 0.0755983430147171

Generator F loss (from generator items example): [0.03121165931224823, 0.024930303916335106]

label_real = [1. 1. 1. 1.]

label_fake = [0. 0. 0. 0.]

scores_realDg = [0.3103015 0.4606359 0.3552073 0.37660494]

scores_fakeDg = [0.09536056 0.13907719 0.01612356 0.07235076]

scores_fakeG = [-0.00598993 0.02842934 -0.03400609 -0.06411391]

scores_realDf = [1.2254677 0.5738747 0.75481224 1.3224378]

scores_fakeDf = [0.48659354 0.38642195 0.489557 0.3740307]

scores_fakeF = [0.48474216 0.4275875 0.4183231 0.8027175]

saving to /content/drive/My Drive/MachineLearning/Model-Epoch/...

Последний log

Epoch number: 70

****Batch number: 200****

Discriminator G loss: **0.94**67716997861862

Generator G loss (from discr + from generat): 0.09722490401938558 + 0.04543204000219703

Generator G loss (from generator items example): [0.028439411893486977, 0.016194166615605354]

Discriminator F loss: 0.9388215026259422

Generator F loss (from discr + from generat): 0.06495836373418569 + 0.042725331336259845

Generator F loss (from generator items example): [0.030994214117527008, 0.018117938190698624]

label_real = [1. 1. 1. 1.]

label_fake = [0. 0. 0. 0.]

scores_realDg = [0.06463666 0.06326077 **0.013**71684 -0.02510134]

scores_fakeDg = [0.01937616 -0.04755492 **0.017**75902 0.00606149]

scores_fakeG = [0.00081293 -0.02301382 -0.03056665 -0.01065779]

scores_realDf = [-0.5296027 -0.5659209 -0.50636953 -0.4858627]

scores_fakeDf = [-0.523952 -0.53265977 -0.49002036 -0.52794844]

scores_fakeF = [-0.5095857 -0.5177655 -0.4713478 -0.53119624]

saving to /content/drive/My Drive/MachineLearning/Model-Epoch/...

Текущий вариант loss-функции (идеи RelativisticGAN)

```
def generator_loss_from_discr(scores_list_real, scores_list_fake, label_real):
    scores = []
    scores_real_mean = torch.mean(torch.stack(scores_list_real))
    scores_fake_mean = torch.mean(torch.stack(scores_list_fake))
    for scores_real, scores_fake in zip(scores_list_real, scores_list_fake):
        scores.append((torch.mean((scores_real - scores_fake_mean) ** 2)      # учим генератор обманывать
                      + torch.mean((scores_fake - scores_real_mean) ** 2))/2)
    return torch.mean(torch.stack(scores))

def discriminator_loss(scores_real, scores_fake, label_real):
    loss1 = torch.mean((scores_real - torch.mean(scores_fake) - label_real) ** 2) # учим дискриминатор
    loss2 = torch.mean((scores_fake - torch.mean(scores_real) + label_real) ** 2)
    return (loss1 + loss2)/2
```

Важный момент: отключаем у дискриминатора на выходе torch.nn.Sigmoid

Что еще проверить

Wasswrshtain gan

<https://machinelearningmastery.com/how-to-implement-wasserstein-loss-for-generative-adversarial-networks/>

CycleGan

Сравнить с <http://shikib.com/CycleGan.html>

<https://machinelearningmastery.com/cyclegan-tutorial-with-keras/>

Fast neural style

<https://github.com/abhiskk/fast-neural-style>

Лучший код - <https://github.com/AlexiaJM/RelativisticGAN>