Autoencoders

Done by Shkarbanenko Mikhail

Stepik User ID: 537953169

Telegram @IInfiniteTsukuyomi

Часть 1. Vanilla Autoencoder (10 баллов)

1.1. Подготовка данных (0.5 балла)

```
import numpy as np
from torch.autograd import Variable
from torchvision import datasets
from torchvision import transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data as data_utils
import torch
import matplotlib.pyplot as plt
%matplotlib inline
from IPython import display

def fetch_dataset(attrs_name = "lfw_attributes.txt",
                      images_name = "lfw-deepfunneled",
                      dx=80,dy=80,
                      dimx=64,dimy=64
    ):

    #download if not exists
    if not os.path.exists(images_name):
        print("images not found, donwloading...")
        os.system("wget http://vis-www.cs.umass.edu/lfw/lfw-
deepfunneled.tgz -O tmp.tgz")
        print("extracting...")
        os.system("tar xvzf tmp.tgz && rm tmp.tgz")
        print("done")
        assert os.path.exists(images_name)

    if not os.path.exists(attrs_name):
        print("attributes not found, downloading...")
        os.system("wget
http://www.cs.columbia.edu/CAVE/databases/pubfig/download/%s" %
attrs_name)
        print("done")
```

```
    #read attrs
    df_attrs = pd.read_csv("lfw_attributes.txt",sep='\t',skiprows=1,)
    df_attrs = pd.DataFrame(df_attrs.iloc[:,:-1].values, columns =
df_attrs.columns[1:])


    #read photos
    photo_ids = []
    for dirpath, dirnames, filenames in os.walk(images_name):
        for fname in filenames:
            if fname.endswith(".jpg"):
                fpath = os.path.join(dirpath,fname)
                photo_id = fname[:-4].replace('_',' ').split()
                person_id = ' '.join(photo_id[:-1])
                photo_number = int(photo_id[-1])

photo_ids.append({'person':person_id,'imagenum':photo_number,'photo_pa
th':fpath})

    photo_ids = pd.DataFrame(photo_ids)
    # print(photo_ids)
    #mass-merge
    #(photos now have same order as attributes)
    df = pd.merge(df_attrs,photo_ids,on=('person','imagenum'))

    assert len(df)==len(df_attrs),"lost some data when merging
dataframes"

    # print(df.shape)
    #image preprocessing
    all_photos =df['photo_path'].apply(skimage.io.imread)\
                                .apply(lambda img:img[dy:-dy,dx:-dx])\
                                .apply(lambda img: resize(img,
[dimx,dimy]))

    all_photos = np.stack(all_photos.values)#.astype('uint8')
    all_attrs = df.drop(["photo_path","person","imagenum"],axis=1)

    return all_photos, all_attrs

# The following line fetches you two datasets: images, usable for
autoencoder training and attributes.
# Those attributes will be required for the final part of the
assignment (applying smiles), so please keep them in mind

import os
import pandas as pd
import skimage.io
from skimage.transform import resize
```

```
data, attrs = fetch_dataset();
```

Разбейте выборку картинок на train и val, выведите несколько картинок в output, чтобы посмотреть, как они выглядят, и приведите картинки к тензорам pytorch, чтобы можно было скормить их сети:

Разбиение датасета с картинками на train/val.

```
from sklearn.model_selection import train_test_split

peoples_faces_data = test_image = torch.permute(torch.Tensor(data),
(0, 3, 1, 2))
X_train_pf, X_val_pf = train_test_split(peoples_faces_data,
test_size=0.2, shuffle=False)

BATCH_SIZE_PF = 32
train_loader_pf = torch.utils.data.DataLoader(X_train_pf,
batch_size=BATCH_SIZE_PF)
val_loader_pf = torch.utils.data.DataLoader(X_val_pf,
batch_size=BATCH_SIZE_PF)

print(X_train_pf.shape)
print(torch.min(X_train_pf), torch.max(X_train_pf))

torch.Size([10514, 3, 64, 64])
tensor(0.) tensor(1.)
```

Функция для визуализации картинок.

```
def imshow(data, n_rows, n_cols, figsize, title=None):
    """
    Show batch of images.
    """
    image_data = data.permute(0, 2, 3, 1)
    fig, axs = plt.subplots(n_rows, n_cols, figsize=figsize)
    for i, ax_i in enumerate(axs.flatten()):
        ax_i.imshow(image_data[i])
    fig.suptitle(title)
    plt.show()
    #plt.close()
    #return fig

imshow(X_train_pf[0:10], 2, 5, (18, 6), title='Картинки из датасета с
лицами людей')
```

1.2. Архитектура модели (1.5 балла)

В этом разделе мы напишем и обучем обычный автоэнкодер.

^ напомню, что автоэнкодер выглядит вот так

```
DIM_CODE_AE = 256 # выберите размер латентного вектора
```

Реализуем autoencoder. Архитектуру (conv, fully-connected, ReLu, etc) можете выбирать сами. Экспериментируйте!

Реализация модели Автоэнкодер.

```
from copy import deepcopy

class Reshape(nn.Module):
    def __init__(self, *args):
        super(Reshape, self).__init__()
        self.shape = args

    def forward(self, x):
        return x.view(self.shape)

class Autoencoder(nn.Module):
    def __init__(self, latent_dim=DIM_CODE_AE):
        super().__init__()

        self.encoder = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=3, padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
```

```
            nn.Flatten(),
            nn.Linear(64*64*64, latent_dim)
        )

        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 64*64*64),
            Reshape(-1, 64, 64, 64),
            nn.ConvTranspose2d(64, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, kernel_size=3, padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.ConvTranspose2d(16, 3, kernel_size=3, padding=1),
            nn.Sigmoid()
        )

    def forward(self, x):
        latent_code = self.encoder(x)
        reconstruction = self.decoder(latent_code)
        return reconstruction, latent_code
```

Инициализация модели Автоэнкодер.

```
DEVICE_AE = 'cuda:0' if torch.cuda.is_available() else 'cpu'
N_EPOCHS_AE = 30

criterion_ae = F.mse_loss
cnn_autoencoder = Autoencoder().to(DEVICE_AE)
cnn_ae_opt = torch.optim.Adam(cnn_autoencoder.parameters(), lr=1e-3,
weight_decay=1e-5)
cnn_ae_sched = torch.optim.lr_scheduler.StepLR(cnn_ae_opt,
step_size=5, gamma=0.8)
```

1.3 Обучение (2 балла)

Осталось написать код обучения автоэнкодера. При этом было бы неплохо в процессе иногда смотреть, как автоэнкодер реконструирует изображения на данном этапе обучения. Наример, после каждой эпохи (прогона train выборки через автоэекодер) можно смотреть, какие реконструкции получились для каких-то изображений val выборки.

А, ну еще было бы неплохо выводить графики train и val лоссов в процессе тренировки =)

Проверка размерностей.

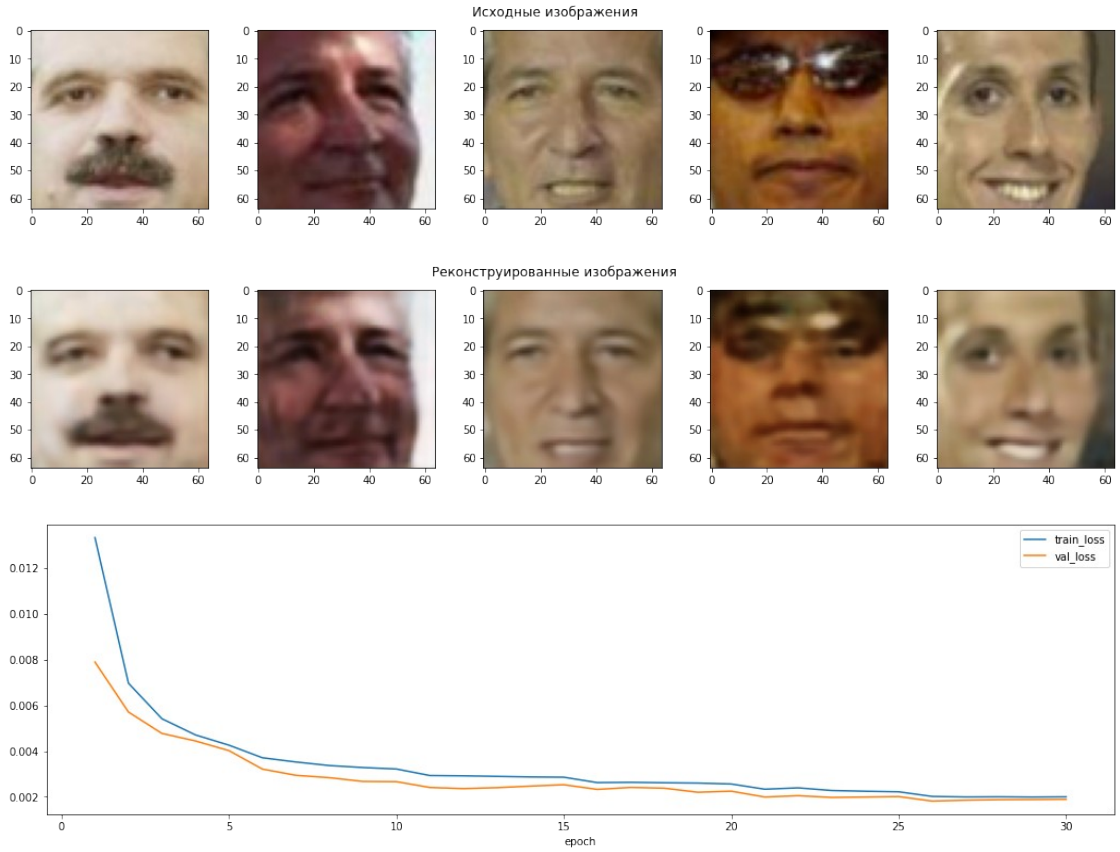
```
def predict_ae(data, model=cnn_autoencoder, device=DEVICE_AE):
    prediction = None
    model.eval()
    with torch.no_grad():
```

```
        prediction = model(data.to(device))
    return prediction

def train_ae(model=cnn_autoencoder, optimizer=cnn_ae_opt,
criterion=criterion_ae, scheduler=cnn_ae_sched,
             device=DEVICE_AE, n_epochs=N_EPOCHS_AE,
             train_loader=train_loader_pf, val_loader=val_loader_pf):

    torch.cuda.empty_cache()
    train_losses = []
    val_losses = []

    for epoch in range(n_epochs):

        torch.cuda.empty_cache()
        model.train()
        train_losses_per_epoch = []
        val_losses_per_epoch = []

        for i, X_batch in enumerate(train_loader):

            optimizer.zero_grad()
            reconstructed = model(X_batch.to(device))[0]
            loss = criterion(reconstructed, X_batch.to(device))
            loss.backward()
            optimizer.step()
            train_losses_per_epoch.append(loss.item())

        train_losses.append(np.mean(train_losses_per_epoch))

        model.eval()
        with torch.no_grad():
            for X_batch in val_loader:
                reconstructed = model(X_batch.to(device))[0]
                loss = criterion(reconstructed, X_batch.to(device))
                val_losses_per_epoch.append(loss.item())

        val_losses.append(np.mean(val_losses_per_epoch))
        scheduler.step()

        # imshow
        display.clear_output(wait=True)
        reconstructed_sample = None
        model.eval()
        with torch.no_grad():
            reconstructed_sample = model(X_val_pf[0:5].to(device))
[0].to('cpu')
        imshow(X_val_pf[0:5], 1, 5, (18, 3), title='Исходные
изображения')
        imshow(reconstructed_sample, 1, 5, (18, 3),
```

```
title='Реконструированные изображения')
        plt.show()

    # loss plot
    plt.figure(figsize=(18, 5))
    plt.plot([epoch+1 for epoch in range(n_epochs)], train_losses,
label='train_loss')
    plt.plot([epoch+1 for epoch in range(n_epochs)], val_losses,
label='val_loss')
    plt.xlabel('epoch')
    plt.legend()
    plt.show()
```

Давайте посмотрим, как наш тренированный автоэекодер кодирует и восстанавливает картинки:

```
train_ae()
```



Not bad, right?

В целом работает неплохо, но некоторые патерны изображений AE не смогу выучить. Например, очки.

1.4. Sampling (2 балла)

Давайте теперь будем не просто брать картинку, прогонять ее через автоэекодер и получать реконструкцию, а попробуем создать что-то НОВОЕ

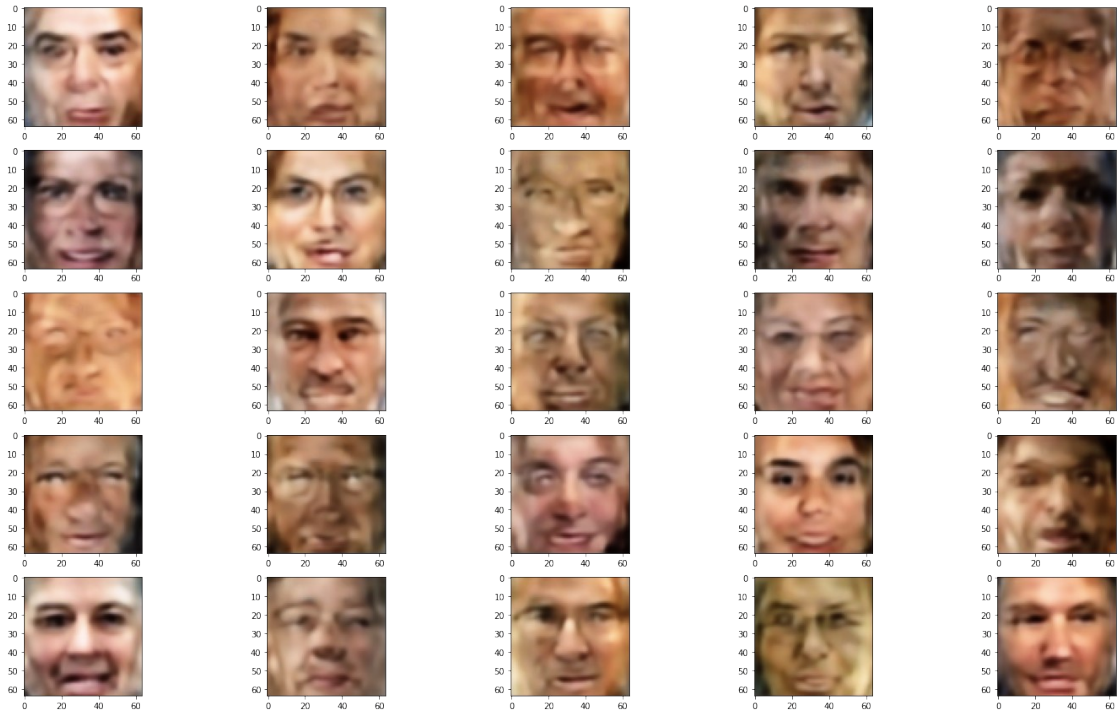
Давайте возьмем и подсунем декодеру какие-нибудь сгенерированные нами векторы (например, из нормального распределения) и посмотрим на результат реконструкции декодера:

_Подсказка:_Е сли вместо лиц у вас выводится непонятно что, попробуйте посмотреть, как выглядят латентные векторы картинок из датасета. Так как в обучении нейронных сетей есть определенная доля рандома, векторы латентного слоя могут быть распределены НЕ как `np.random.randn(25, <latent_space_dim>)`. А чтобы у нас получались лица при запихивании вектора декодеру, вектор должен быть распределен так же, как латентные векторы реальных фоток. Так что в таком случае придется рандом немного подогнать.

Генерация изображений. Вектора сэмплирую из нормального распределения с параметрами mu и sigma из валидационного латентного пространства.

```
# сгенерируем 25 рандомных векторов размера latent_space
latent_mu = predict_ae(X_val_pf)[1].to('cpu').mean(0)
latent_sigma = predict_ae(X_val_pf)[1].to('cpu').std(0)
z = latent_sigma * torch.randn(25, DIM_CODE_AE) + latent_mu
generated_images = None
cnn_autoencoder.eval()
with torch.no_grad():
        generated_images =
cnn_autoencoder.decoder(z.to(DEVICE_AE)).to('cpu')
imshow(generated_images, 5, 5, (25, 15), title='Сгенерированные
изображения')
```

Сгенерированные изображения



Time to make fun! (4 балла)

Давайте научимся пририсовывать людям улыбки =)

План такой:

1. Нужно выделить "вектор улыбки": для этого нужно из выборки изображений найти несколько (~15) людей с улыбками и столько же без.

Найти людей с улыбками вам поможет файл с описанием датасета, скачанный вместе с датасетом. В нем указаны имена картинок и присутствубщие атрибуты (улыбки, очки...)

1. Вычислить латентный вектор для всех улыбающихся людей (прогнать их через encoder) и то же для всех грустненьких

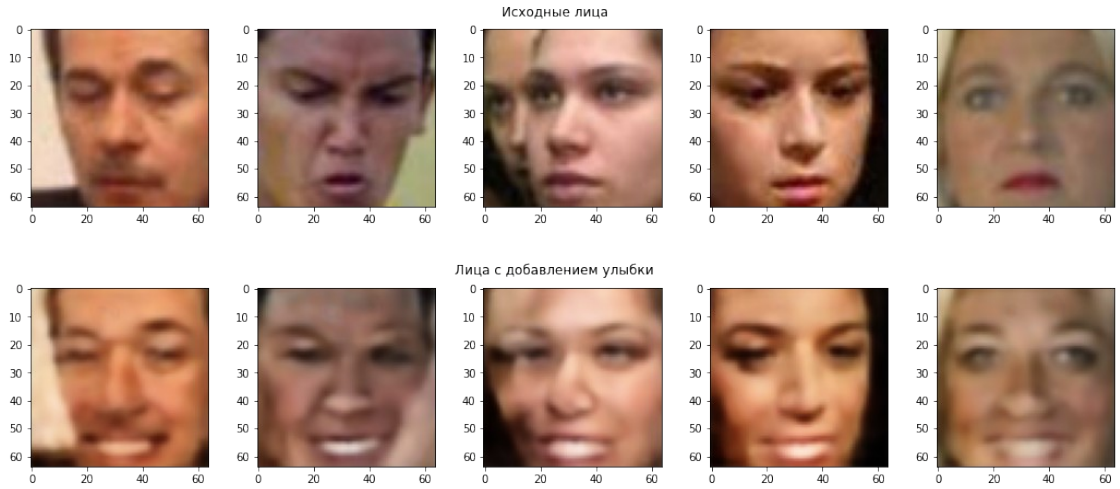
2. Вычислить, собственно, вектор улыбки -- посчитать разность между средним латентным вектором улыбающихся людей и средним латентным вектором грустных людей

3. А теперь приделаем улыбку грустному человеку: добавим полученный в пункте 3 вектор к латентному вектору грустного

человека и прогоним полученный вектор через decoder. Получим того же человека, но уже не грустненького!

Добавление грустным лицам улыбок.

```
smiling_people = X_train_pf[attrs[attrs['Smiling'] >
2].head(15).index]
sad_people = X_train_pf[attrs[attrs['Smiling'] < - 2].head(15).index]

smiling_people_encoded =
cnn_autoencoder.encoder(smiling_people.to(DEVICE_AE))
sad_people_encoded = cnn_autoencoder.encoder(sad_people.to(DEVICE_AE))
diff = (smiling_people_encoded - sad_people_encoded).mean(0)
generated_smiling_people_encoded = sad_people_encoded + diff
generated_smiling_people =
cnn_autoencoder.decoder(generated_smiling_people_encoded).detach().to(
'cpu')

imshow(sad_people[0:5], 1, 5, (18, 3), title='Исходные лица')
imshow(generated_smiling_people[0:5], 1, 5, (18, 3), title='Лица с
добавлением улыбки')
```



Вуаля! Вы восхитительны!

Теперь вы можете пририсовывать людям не только улыбки, но и много чего другого -- закрывать/открывать глаза, пририсовывать очки... в общем, все, на что хватит фантазии и на что есть атрибуты в `all_attrs`:)

Часть 2: Variational Autoencoder (10 баллов)

Займемся обучением вариационных автоэнкодеров — проапгрейженной версии AE. Обучать будем на датасете MNIST, содержащем написанные от руки цифры от 0 до 9

```
BATCH_SIZE_MNIST = 32

# MNIST Dataset
train_dataset_mnist = datasets.MNIST(root='./mnist_data/', train=True,
transform=transforms.ToTensor(), download=True)
val_dataset_mnist = datasets.MNIST(root='./mnist_data/', train=False,
transform=transforms.ToTensor(), download=False)

# Data Loader (Input Pipeline)
train_loader_mnist =
torch.utils.data.DataLoader(dataset=train_dataset_mnist,
batch_size=BATCH_SIZE_MNIST, shuffle=True)
val_loader_mnist =
torch.utils.data.DataLoader(dataset=val_dataset_mnist,
batch_size=BATCH_SIZE_MNIST, shuffle=False)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-
ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-
ubyte.gz to ./mnist_data/MNIST/raw/train-images-idx3-ubyte.gz
```


```
{"version_major":2,"version_minor":0,"model_id":"efd94cf829614514971fe
99e9acbe107"}
```


```
Extracting ./mnist_data/MNIST/raw/train-images-idx3-ubyte.gz to
./mnist_data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-
ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-
ubyte.gz to ./mnist_data/MNIST/raw/train-labels-idx1-ubyte.gz
```


```
{"version_major":2,"version_minor":0,"model_id":"21b9ebaeac5d44a6893b9
305a431d631"}
```


```
Extracting ./mnist_data/MNIST/raw/train-labels-idx1-ubyte.gz to
./mnist_data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
to ./mnist_data/MNIST/raw/t10k-images-idx3-ubyte.gz
```


```
{"version_major":2,"version_minor":0,"model_id":"205ae9a177394a22b7f9f
9f911092d60"}
```


```
Extracting ./mnist_data/MNIST/raw/t10k-images-idx3-ubyte.gz to
./mnist_data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
to ./mnist_data/MNIST/raw/t10k-labels-idx1-ubyte.gz
```

{"version_major":2,"version_minor":0,"model_id":"94c8b623e89f4cf68fd4ceeee30186bc"}

```
Extracting ./mnist_data/MNIST/raw/t10k-labels-idx1-ubyte.gz to
./mnist_data/MNIST/raw
```

2.1 Архитектура модели и обучение (2 балла)

Реализуем VAE. Архитектуру (conv, fully-connected, ReLu, etc) можете выбирать сами. Рекомендуем пользоваться более сложными моделями, чем та, что была на семинаре:) Экспериментируйте!

Реализация VAE.

```
class VAE(nn.Module):

    def __init__(self, latent_dim=128):
        super(VAE, self).__init__()

        self.encoder = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.Flatten()
        )

        self.mu = nn.Linear(64*28*28, latent_dim)
        self.logsigma = nn.Linear(64*28*28, latent_dim)

        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 64*28*28),
            Reshape(-1, 64, 28, 28),
            nn.ConvTranspose2d(64, 32, kernel_size=3, stride=1,
padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, kernel_size=3, stride=1,
padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.ConvTranspose2d(16, 1, kernel_size=3, stride=1,
padding=1),
            nn.Sigmoid()
        )
```

```
    def encode(self, x):
        encoded = self.encoder(x)
        mu = self.mu(encoded)
        logsigma = self.logsigma(encoded)
        return mu, logsigma

    def gaussian_sampler(self, mu, logsigma):
        if self.training:
            std = torch.exp(0.5 * logsigma)
            eps = torch.randn_like(std)
            return mu + (eps * std)
        else:
            return mu

    def decode(self, z):
        reconstruction = self.decoder(z)
        return reconstruction

    def forward(self, x):
        mu, logsigma = self.encode(x)
        z = self.gaussian_sampler(mu, logsigma)
        reconstruction = self.decode(z)
        return mu, logsigma, reconstruction
```

Определим лосс и его компоненты для VAE:

Надеюсь, вы уже прочитали материал в towardsdatascience (или еще где-то) про VAE и знаете, что лосс у VAE состоит из двух частей: KL и log-likelihood.

Общий лосс будет выглядеть так:

$$L = -D_{KL}¿$$

Формула для KL-дивергенции:

$$D_{KL} = -\frac{1}{2}\sum_{i=1}^{dimZ}\left(1 + log\left(\sigma_i^2\right) - \mu_i^2 - \sigma_i^2\right)$$

В качестве log-likelihood возьмем привычную нам кросс-энтропию.

Реализация лосс функции для VAE.

```
def KL_divergence(mu, logsigma):
    """
    часть функции потерь, которая отвечает за "близость" латентных
представлений разных людей
    """
    loss = -0.5 * torch.sum(1 + logsigma - mu ** 2 - logsigma.exp())
    return loss

def log_likelihood(x, reconstruction):
```

```
    """
    часть функции потерь, которая отвечает за качество реконструкции
(как mse в обычном autoencoder)
    """
    loss = nn.BCELoss(reduction='sum')
    return loss(reconstruction, x)

def loss_vae(x, mu, logsigma, reconstruction):
    return KL_divergence(mu, logsigma) + log_likelihood(x,
reconstruction)
```

Инициализация и обучения модели.

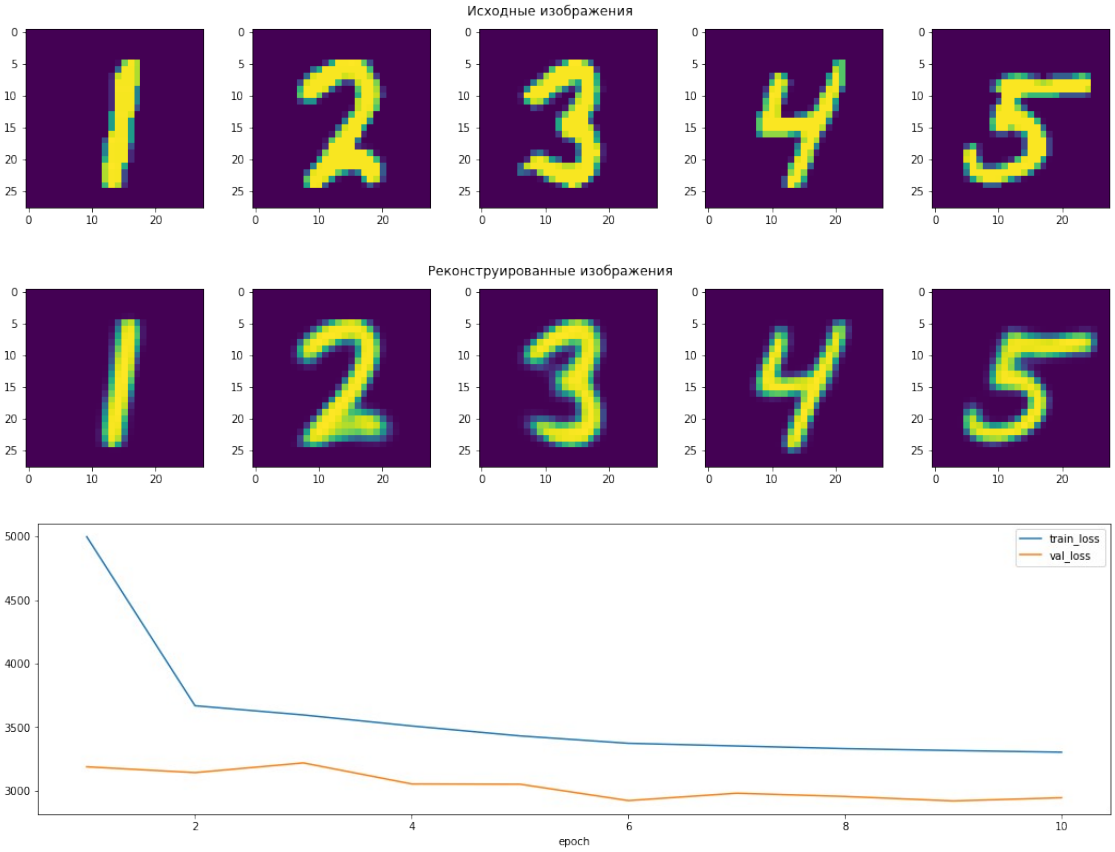
И обучим модель:

```
DEVICE_VAE = 'cuda:0' if torch.cuda.is_available() else 'cpu'
N_EPOCHS_VAE = 10

vae_loss = loss_vae
vae = VAE().to(DEVICE_VAE)
vae_opt = torch.optim.Adam(vae.parameters(), lr=1e-3, weight_decay=1e-
5)
vae_sched = torch.optim.lr_scheduler.StepLR(vae_opt, step_size=5,
gamma=0.8)

def predict_vae(data, model=vae, device=DEVICE_VAE):
    prediction = None
    model.eval()
    with torch.no_grad():
        mu, logsigma, reconstruction = model(data.to(device))
    return mu, logsigma, reconstruction

def train_vae(model=vae, optimizer=vae_opt, criterion=vae_loss,
scheduler=vae_sched,
              device=DEVICE_VAE, n_epochs=N_EPOCHS_VAE,
              train_loader=train_loader_mnist,
val_loader=val_loader_mnist):

    torch.cuda.empty_cache()
    train_losses = []
    val_losses = []

    for epoch in range(n_epochs):

        torch.cuda.empty_cache()
        model.train()
        train_losses_per_epoch = []
        val_losses_per_epoch = []

        for X_batch, _ in train_loader:
```

```
            optimizer.zero_grad()
            mu, logsigma, reconstruction = model(X_batch.to(device))
            loss = criterion(X_batch.to(device), mu, logsigma,
reconstruction)
            loss.backward()
            optimizer.step()
            train_losses_per_epoch.append(loss.item())

        train_losses.append(np.mean(train_losses_per_epoch))

        model.eval()
        with torch.no_grad():
            for X_batch, _ in val_loader:
                mu, logsigma, reconstruction =
model(X_batch.to(device))
                loss = criterion(X_batch.to(device), mu, logsigma,
reconstruction)
                val_losses_per_epoch.append(loss.item())

        val_losses.append(np.mean(val_losses_per_epoch))
        scheduler.step()

        # imshow
        display.clear_output(wait=True)
        reconstructed_sample = None
        model.eval()
        with torch.no_grad():
            reconstructed_sample = model(X_batch[0:5].to(device))
[2].to('cpu')
        imshow(X_batch[0:5], 1, 5, (18, 3), title='Исходные
изображения')
        imshow(reconstructed_sample, 1, 5, (18, 3),
title='Реконструированные изображения')
        plt.show()

    # loss plot
    plt.figure(figsize=(18, 5))
    plt.plot([epoch+1 for epoch in range(n_epochs)], train_losses,
label='train_loss')
    plt.plot([epoch+1 for epoch in range(n_epochs)], val_losses,
label='val_loss')
    plt.xlabel('epoch')
    plt.legend()
    plt.show()
```

Давайте посмотрим, как наш тренированный VAE кодирует и восстанавливает картинки:

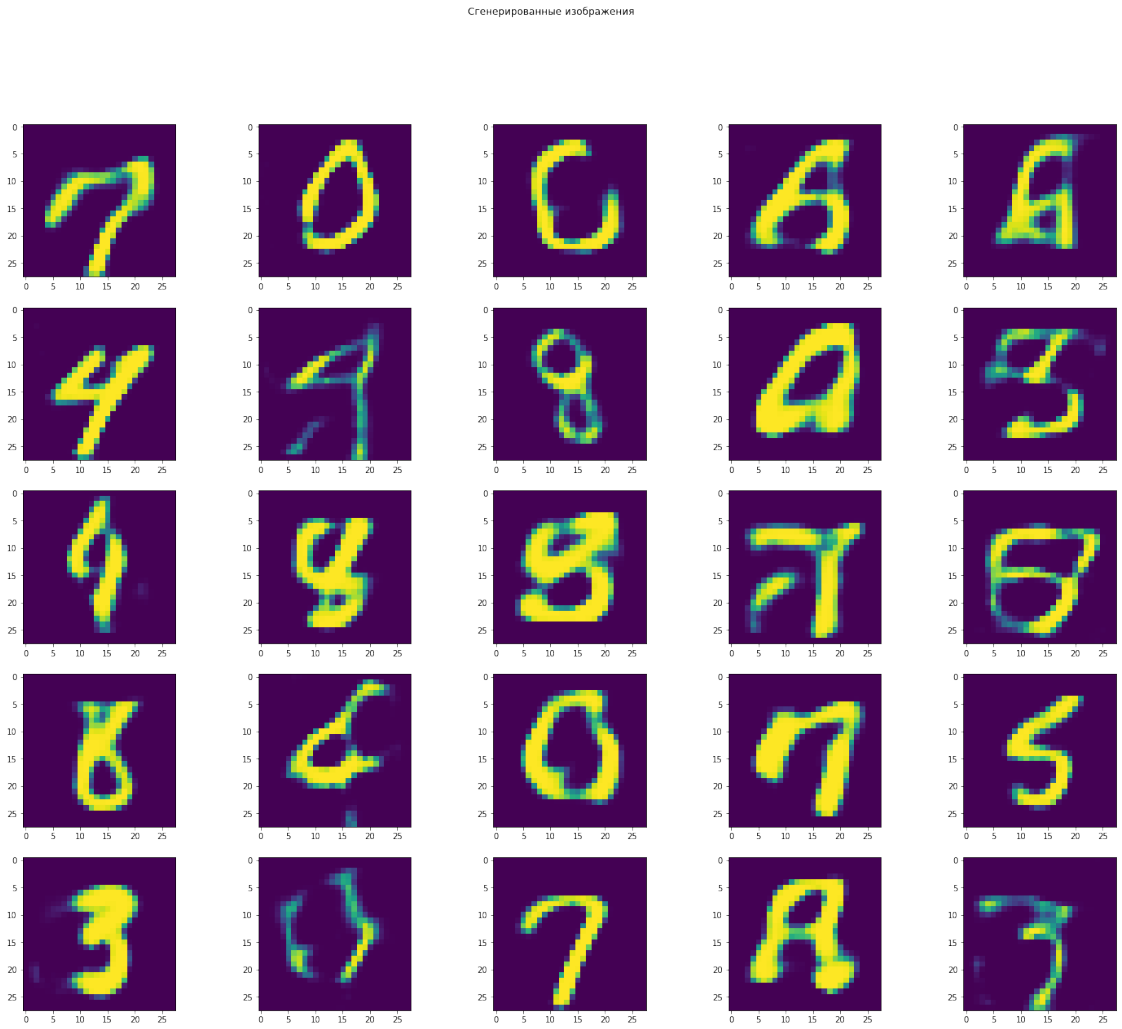
```
train_vae()
```



Давайте попробуем проделать для VAE то же, что и с обычным автоэнкодером -- подсунуть decoder'у из VAE случайные векторы из нормального распределения и посмотреть, какие картинки получаются:

Генерация изображений.

```
X_train_mnist = torch.Tensor([i[0].tolist() for i in
train_dataset_mnist])
y_train_mnist = torch.Tensor([i[1] for i in train_dataset_mnist])
X_val_mnist = torch.Tensor([i[0].tolist() for i in val_dataset_mnist])
y_val_mnist = torch.Tensor([i[1] for i in val_dataset_mnist])

latent_mu = predict_vae(X_train_mnist[:10000])[0].to('cpu').mean(0)
latent_sigma = predict_vae(X_train_mnist[:10000])[1].to('cpu').mean(0)
z = latent_sigma * torch.randn(25, 128) + latent_mu
generated_images = None
vae.eval()
with torch.no_grad():
        generated_images = vae.decoder(z.to(DEVICE_VAE)).to('cpu')
imshow(generated_images, 5, 5, (25, 20), title='Сгенерированные
изображения')
torch.cuda.empty_cache()
```



2.2. Latent Representation (2 балла)

Давайте посмотрим, как латентные векторы картинок лиц выглядят в пространстве. Ваша задача -- изобразить латентные векторы картинок точками в двумерном просторанстве.

Это позволит оценить, насколько плотно распределены латентные векторы изображений цифр в пространстве.

Плюс давайте сделаем такую вещь: покрасим точки, которые соответствуют картинкам каждой цифры, в свой отдельный цвет

Подсказка: красить -- это просто =) У plt.scatter есть параметр c (color), см. в документации.

Итак, план:

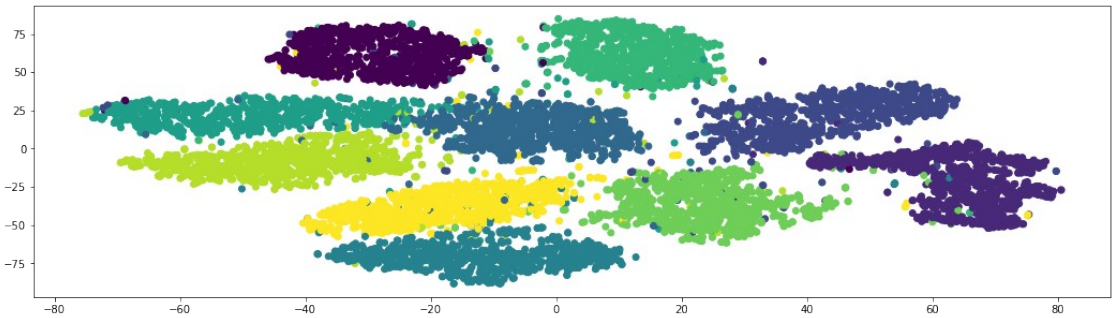
1. Получить латентные представления картинок тестового датасета

2. С помощтю TSNE (есть в sklearn) сжать эти представления до размерности 2 (чтобы можно было их визуализировать точками в пространстве)
3. Визуализировать полученные двумерные представления с помощью matplotlib.scatter, покрасить разными цветами точки, соответствующие картинкам разных цифр.

```
from sklearn.manifold import TSNE

X_val_mnist_enc = []
vae.eval()
with torch.no_grad():
    for X_batch in X_val_mnist:
        mu, logsigma = vae.encode(X_batch.unsqueeze(0).to(DEVICE_VAE))
        latent_vec = vae.gaussian_sampler(mu, logsigma).to('cpu')
        X_val_mnist_enc.append(latent_vec.tolist())

X_val_mnist_enc = torch.Tensor(X_val_mnist_enc).squeeze().numpy()

X_val_mnist_enc_compressed =
TSNE(n_components=2).fit_transform(X_val_mnist_enc)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/manifold/_t_sne.py:783:
FutureWarning: The default initialization in TSNE will change from
'random' to 'pca' in 1.2.
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/manifold/_t_sne.py:793:
FutureWarning: The default learning rate in TSNE will change from
200.0 to 'auto' in 1.2.
  FutureWarning,
```

```
plt.figure(figsize=(18, 5))
plt.scatter(X_val_mnist_enc_compressed[:, 0],
X_val_mnist_enc_compressed[:, 1], c=val_dataset_mnist.targets)
plt.show()
```



Что вы думаете о виде латентного представления?

Цифры разделились по кластерам в латентном пространстве. Распределения векторов в каждом кластере имеют не слишком большую дисперсию.

Congrats v2.0!

2.3. Conditional VAE (6 баллов)

Мы уже научились обучать обычный AE на датасете картинок и получать новые картинки, используя генерацию шума и декодер. Давайте теперь допустим, что мы обучили AE на датасете MNIST и теперь хотим генерировать новые картинки с числами с помощью декодера (как выше мы генерили рандомные лица). И вот нам понадобилось сгенерировать цифру 8, и мы подставляем разные варианты шума, но восьмерка никак не генерится:(

Хотелось бы добавить к нашему AE функцию "выдай мне рандомное число из вот этого вот класса", где классов десять (цифры от 0 до 9 образуют десять классов). Conditional AE — так называется вид автоэнкодера, который предоставляет такую возможность. Ну, название "conditional" уже говорит само за себя.

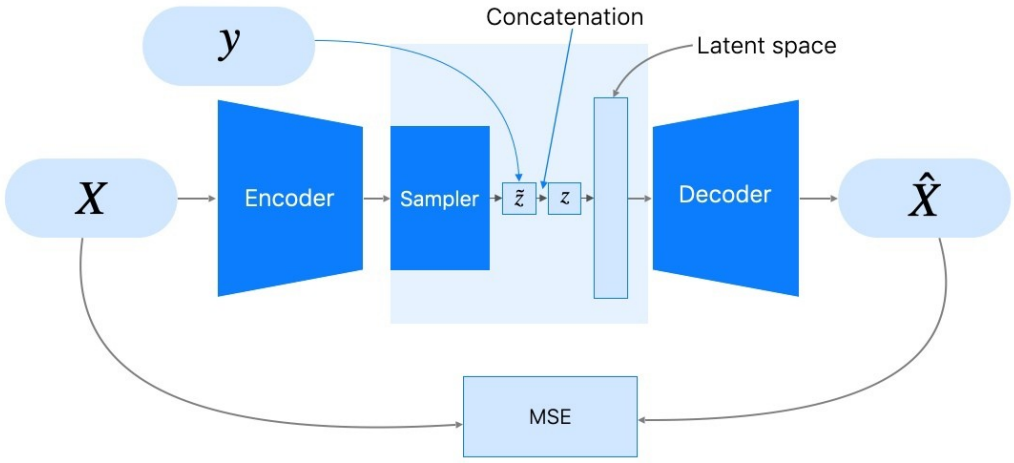
И в этой части задания мы научимся такие обучать.

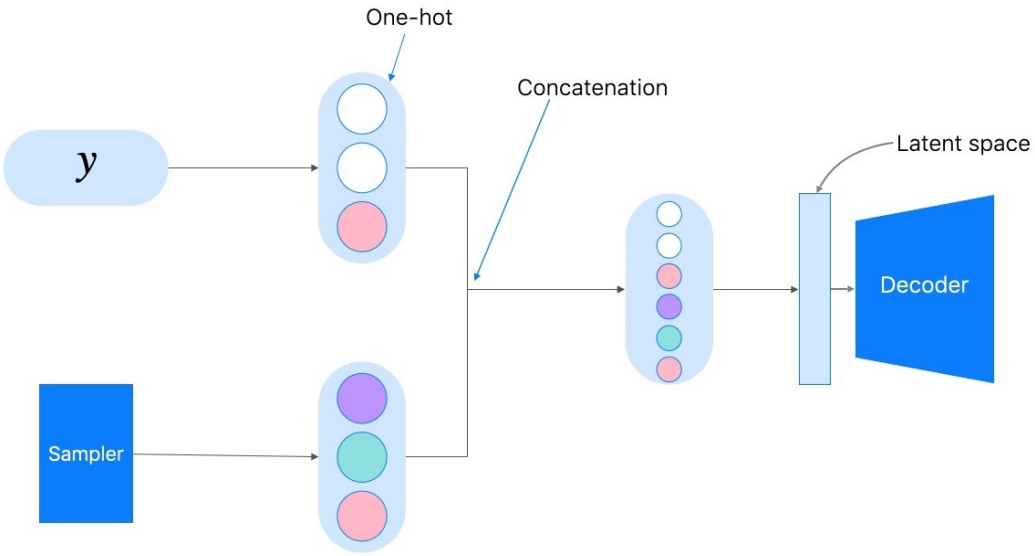
Архитектура

На картинке ниже представлена архитектура простого Conditional VAE.

По сути, единственное отличие от обычного -- это то, что мы вместе с картинкой в первом слое энкодера и декодера передаем еще информацию о классе картинки.

То есть, в первый (входной) слой энкодера подается конкатенация картинки и информации о классе (например, вектора из девяти нулей и одной единицы). В первый слой декодера подается конкатенация латентного вектора и информации о классе.





На всякий случай: это VAE, то есть, latent у него все еще состоит из mu и sigma

Таким образом, при генерации новой рандомной картинки мы должны будем передать декодеру сконкатенированные латентный вектор и класс картинки.

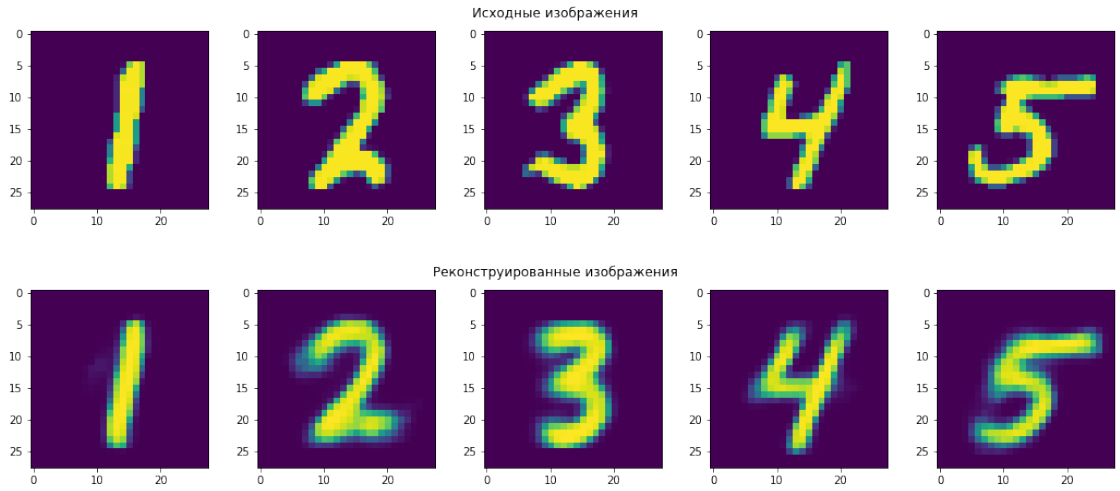
P.S. Также можно передавать класс картинки не только в первый слой, но и в каждый слой сети. То есть на каждом слое конкатенировать выход из предыдущего слоя и информацию о классе.

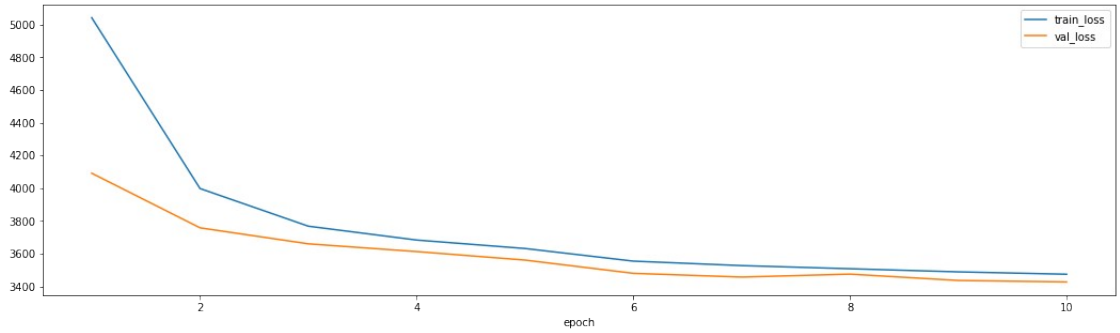
Реализация и обучение CVAE.

```
class CVAE(nn.Module):
    def __init__(self, latent_dim=128):
        super().__init__()

        self.enc_flatten = nn.Flatten()
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28 + 10, 2048),
            nn.ReLU(),
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
        )

        self.mu = nn.Linear(512, latent_dim)
        self.logsigma = nn.Linear(512, latent_dim)

        self.decoder = nn.Sequential(
            nn.Linear(latent_dim + 10, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, 2048),
            nn.ReLU(),
            nn.Linear(2048, 28 * 28)
        )

    def encode(self, x, y):
        flatten = self.enc_flatten(x)
        concat = torch.cat((flatten, y), dim=1)
        encoded = self.encoder(concat)
        mu = self.mu(encoded)
        logsigma = self.logsigma(encoded)
        return mu, logsigma

    def gaussian_sampler(self, mu, logsigma):
        if self.training:
            std = torch.exp(0.5 * logsigma)
            eps = torch.randn_like(std)
            return mu + (eps * std)
        else:
            return mu

    def decode(self, z, y):
        reconstruction = torch.cat((z, y), dim=1)
        reconstruction = self.decoder(reconstruction)
        reconstruction = torch.sigmoid(reconstruction)
        reconstruction = reconstruction.view(-1, 1, 28, 28)
        return reconstruction
```

```
    def forward(self, x, label):
        y = nn.functional.one_hot(label, 10)
        mu, logsigma = self.encode(x, y)
        z = self.gaussian_sampler(mu, logsigma)
        reconstruction = self.decode(z, y)
        return mu, logsigma, reconstruction

DEVICE_CVAE = 'cuda:0' if torch.cuda.is_available() else 'cpu'
N_EPOCHS_CVAE = 10

cvae_loss = loss_vae
cvae = CVAE().to(DEVICE_CVAE)
cvae_opt = torch.optim.Adam(cvae.parameters(), lr=1e-3,
weight_decay=1e-5)
cvae_sched = torch.optim.lr_scheduler.StepLR(cvae_opt, step_size=5,
gamma=0.8)

def train_cvae(model=cvae, optimizer=cvae_opt, criterion=cvae_loss,
scheduler=cvae_sched,
               device=DEVICE_CVAE, n_epochs=N_EPOCHS_CVAE,
               train_loader=train_loader_mnist,
val_loader=val_loader_mnist):

    torch.cuda.empty_cache()
    train_losses = []
    val_losses = []

    for epoch in range(n_epochs):

        torch.cuda.empty_cache()
        model.train()
        train_losses_per_epoch = []
        val_losses_per_epoch = []

        for X_batch, Y_batch in train_loader:

            X_batch = X_batch.to(device)
            Y_batch = Y_batch.to(device)

            optimizer.zero_grad()
            mu, logsigma, reconstruction = model(X_batch, Y_batch)
            loss = criterion(X_batch, mu, logsigma, reconstruction)
            loss.backward()
            optimizer.step()
            train_losses_per_epoch.append(loss.item())

        train_losses.append(np.mean(train_losses_per_epoch))

        model.eval()
        with torch.no_grad():
```

```
            for X_batch, Y_batch in val_loader:
                X_batch = X_batch.to(device)
                Y_batch = Y_batch.to(device)
                mu, logsigma, reconstruction = model(X_batch, Y_batch)
                loss = criterion(X_batch, mu, logsigma,
reconstruction)
                val_losses_per_epoch.append(loss.item())

        val_losses.append(np.mean(val_losses_per_epoch))
        scheduler.step()

        # imshow
        display.clear_output(wait=True)
        reconstructed_sample = None
        model.eval()
        with torch.no_grad():
            reconstructed_sample = model(X_batch[0:5], Y_batch[0:5])
[2].to('cpu')
        imshow(X_batch[0:5].cpu(), 1, 5, (18, 3), title='Исходные
изображения')
        imshow(reconstructed_sample, 1, 5, (18, 3),
title='Реконструированные изображения')
        plt.show()

    # loss plot
    plt.figure(figsize=(18, 5))
    plt.plot([epoch+1 for epoch in range(n_epochs)], train_losses,
label='train_loss')
    plt.plot([epoch+1 for epoch in range(n_epochs)], val_losses,
label='val_loss')
    plt.xlabel('epoch')
    plt.legend()
    plt.show()

train_cvae()
```





Sampling

Тут мы будем сэмплировать из CVAE. Это прикольнее, чем сэмплировать из простого AE/VAE: тут можно взять один и тот же латентный вектор и попросить CVAE восстановить из него картинки разных классов! Для MNIST вы можете попросить CVAE восстановить из одного латентного вектора, например, картинки цифры 5 и 7.

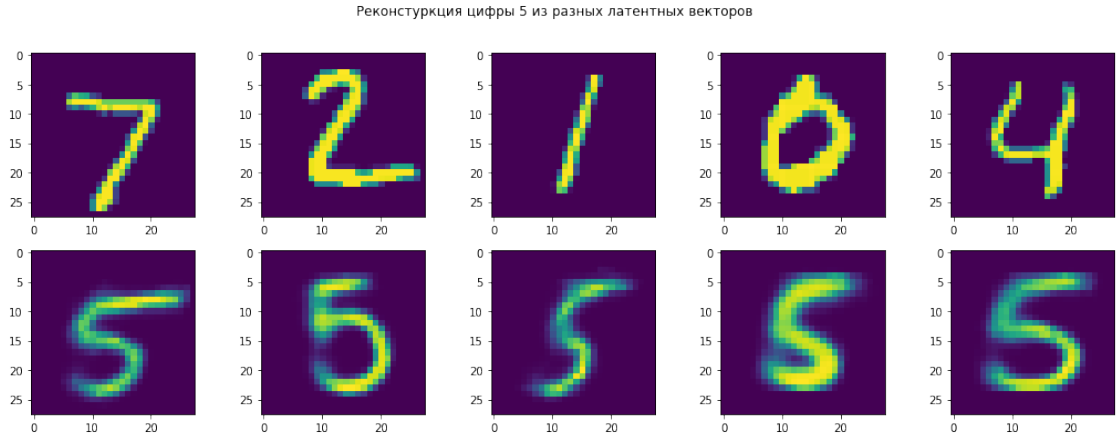
Сэмплинг.

```
def predict_cvae(model=cvae, val_loader=val_loader_mnist,
device=DEVICE_CVAE):
    mu_list, logsigma_list, rec_list = [], [], []
    cvae.eval()
    with torch.no_grad():
        for X_batch, Y_batch in val_loader:
            X_batch = X_batch.to(device)
            Y_batch = Y_batch.to(device)
            mu, logsigma, reconstructed = cvae(X_batch, Y_batch)
            mu_list.append(mu.cpu())
            logsigma_list.append(logsigma.cpu())
            rec_list.append(reconstructed.cpu())
    return torch.cat(mu_list), torch.cat(logsigma_list),
torch.cat(rec_list)
```

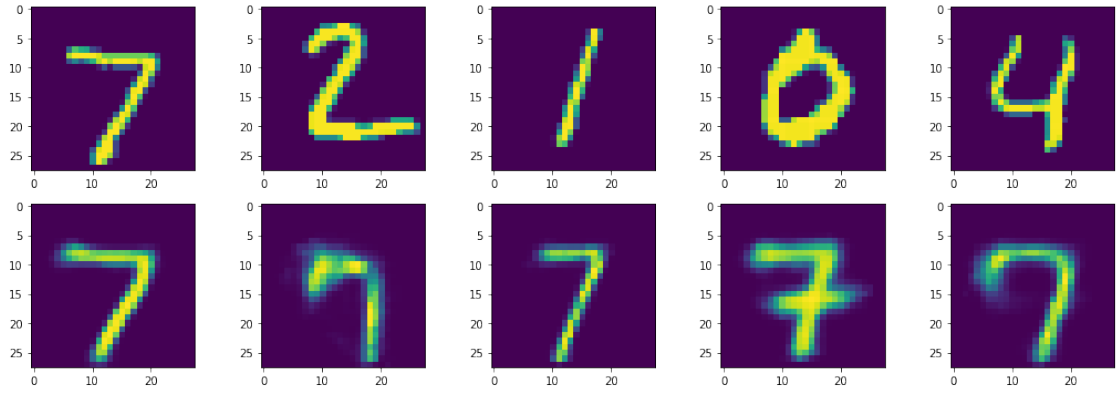
Реконструкция 5, 7 и 9.

```
mu_tensor, logsigma_tensor, rec_tensor= predict_cvae(cvae,
val_loader_mnist)
real_images = next(iter(val_loader_mnist))[0]
label = F.one_hot(torch.LongTensor([5]), 10).view(1, -1)

cvae.eval()
with torch.no_grad():
    rec_tensor = np.array([cvae.decode(mu_tensor[i].view(1, -
1).to('cuda'), label.to('cuda')).cpu().numpy() for i in
range(5)]).squeeze()
    rec_tensor = torch.from_numpy(rec_tensor).unsqueeze(1)

five_generated_data = torch.cat((real_images[:5], rec_tensor), dim=0)
```

```
imshow(five_generated_data, 2, 5, (18, 6), title="Реконстуркция цифры
5 из разных латентных векторов")
```

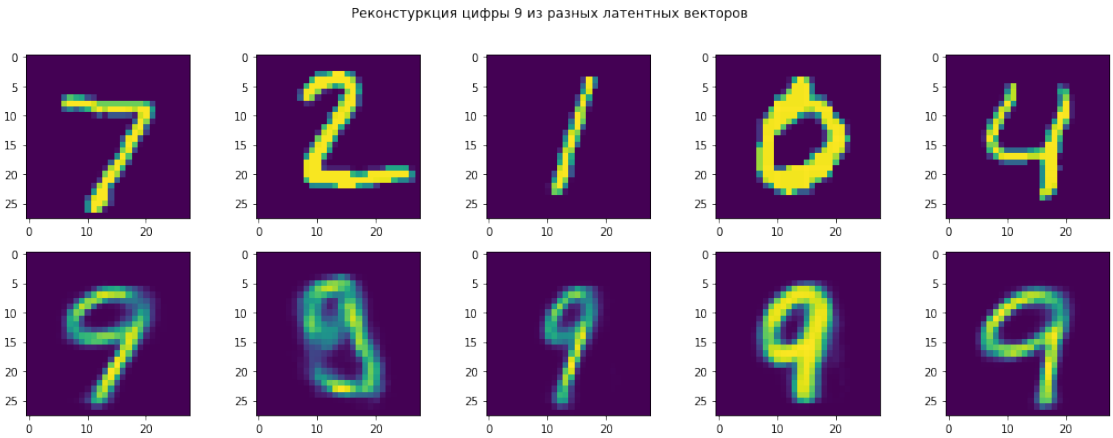



```
mu_tensor, logsigma_tensor, rec_tensor= predict_cvae(cvae,
val_loader_mnist)
real_images = next(iter(val_loader_mnist))[0]
label = F.one_hot(torch.LongTensor([7]), 10).view(1, -1)

cvae.eval()
with torch.no_grad():
    rec_tensor = np.array([cvae.decode(mu_tensor[i].view(1, -
1).to('cuda'), label.to('cuda')).cpu().numpy() for i in
range(5)]).squeeze()
    rec_tensor = torch.from_numpy(rec_tensor).unsqueeze(1)

five_generated_data = torch.cat((real_images[:5], rec_tensor), dim=0)
imshow(five_generated_data, 2, 5, (18, 6), title="Реконстуркция цифры
7 из разных латентных векторов")
```




```
mu_tensor, logsigma_tensor, rec_tensor= predict_cvae(cvae,
val_loader_mnist)
real_images = next(iter(val_loader_mnist))[0]
label = F.one_hot(torch.LongTensor([9]), 10).view(1, -1)
```

```
cvae.eval()
with torch.no_grad():
    rec_tensor = np.array([cvae.decode(mu_tensor[i].view(1, -
1).to('cuda'), label.to('cuda')).cpu().numpy() for i in
range(5)]).squeeze()
    rec_tensor = torch.from_numpy(rec_tensor).unsqueeze(1)

five_generated_data = torch.cat((real_images[:5], rec_tensor), dim=0)
imshow(five_generated_data, 2, 5, (18, 6), title="Реконстуркция цифры
9 из разных латентных векторов")
```



Splendid! Вы великолепны!

Latent Representations

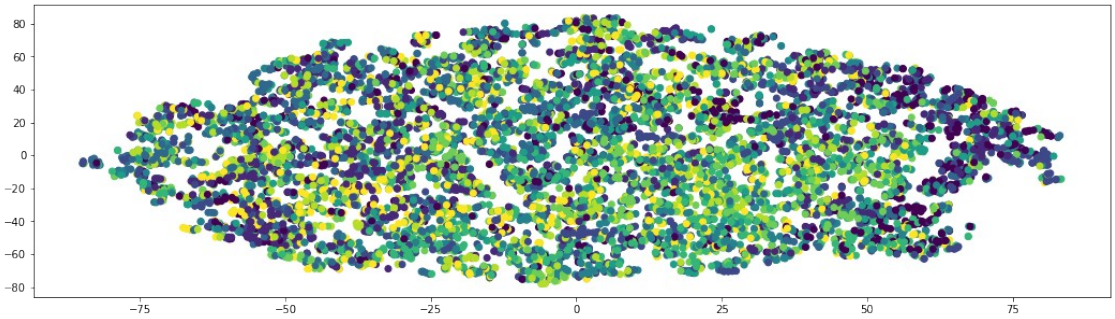
Давайте посмотрим, как выглядит латентное пространство картинок в CVAE и сравним с картинкой для VAE =)

Опять же, нужно покрасить точки в разные цвета в зависимости от класса.

```
mu_tensor_reduced =
torch.from_numpy(TSNE(n_components=2).fit_transform(mu_tensor))
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/manifold/_t_sne.py:783:
FutureWarning: The default initialization in TSNE will change from
'random' to 'pca' in 1.2.
  FutureWarning,
/opt/conda/lib/python3.7/site-packages/sklearn/manifold/_t_sne.py:793:
FutureWarning: The default learning rate in TSNE will change from
200.0 to 'auto' in 1.2.
  FutureWarning,
```

```
plt.figure(figsize=(18, 5))
plt.scatter(mu_tensor_reduced[:, 0], mu_tensor_reduced[:, 1],
c=val_dataset_mnist.targets)
plt.show()
```



По сравнение с обычным VAE спроецированные с помощью TSNE латентные векторы из CVAE получились перемешанными друг с другом (добавление информации о классе видимо не дает сформировать кластеры).

Что вы думаете насчет этой картинки? Отличается от картинки для VAE?

CVAE не формирует кластера из латентных векторов. Я думаю, что это связано с тем, что модели не нужно самой распознавать чем одна цифра отличается от другой, потому что ей на вход уже подана информация о классе, поэтому CVAE располагает вектора другим, вероятно более удобным для него образом.

BONUS 1: Denoising

Внимание! За бонусы доп. баллы не ставятся, но вы можете сделать их для себя.

У автоэнкодеров, кроме сжатия и генерации изображений, есть другие практические применения. Про одно из них эта бонусная часть задания.

Автоэнкодеры могут быть использованы для избавления от шума на фотографиях (denoising). Для этого их нужно обучить специальным образом: input картинка будет зашумленной, а выдавать автоэнкодер должен будет картинку без шума. То есть, loss-функция AE останется той же (MSE между реальной картинкой и выданной), а на вход автоэнкодеру будет подаваться зашумленная картинка.

Для этого нужно взять ваш любимый датасет (датасет лиц из первой части этого задания или любой другой) и сделать копию этого датасета с шумом.

В питоне шум можно добавить так:

```
noise_factor = 0.5
X_noisy = X + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=X.shape)

<тут ваш код обучения автоэнкодера на зашумленных картинках. Не
забудтье разбить на train/test!>

<тут проверка, как AE убирает щум с тестовых картинок. Надеюсь, все
получилось =)>
```

BONUS 2: Image Retrieval

Внимание! За бонусы доп. баллы не ставятся, но вы можете сделать их для себя.

Давайте представим, что весь наш тренировочный датасет -- это большая база данных людей. И вот мы получили картинку лица какого-то человека с уличной камеры наблюдения (у нас это картинка из тестового датасета) и хотим понять, что это за человек. Что нам делать? Правильно -- берем наш VAE, кодируем картинку в латентное представление и ищем среди латентныз представлений лиц нашей базы самые ближайшие!

План:

1. Получаем латентные представления всех лиц тренировочного датасета
2. Обучаем на них LSHForest (`sklearn.neighbors.LSHForest`), например, с `n_estimators=50`
3. Берем картинку из тестового датасета, с помощью VAE получаем ее латентный вектор
4. Ищем с помощью обученного LSHForest ближайшие из латентных представлений тренировочной базы
5. Находим лица тренировочного датасета, которым соответствуют ближайшие латентные представления, визуализируем!

Немного кода вам в помощь: (feel free to delete everything and write your own)

```
codes = <поучите латентные представления картинок из трейна>

# обучаем LSHForest
from sklearn.neighbors import LSHForest
lshf = LSHForest(n_estimators=50).fit(codes)

def get_similar(image, n_neighbors=5):
  # функция, которая берет тестовый image и с помощью метода
kneighbours у LSHForest ищет ближайшие векторы
  # прогоняет векторы через декодер и получает картинки ближайших
людей
```

```
  code = <получение латентного представления image>

  (distances,),(idx,) = lshf.kneighbors(code, n_neighbors=n_neighbors)

  return distances, X_train[idx]

def show_similar(image):

  # функция, которая принимает тестовый image, ищет ближайшие к нему и
визуализирует результат

    distances,neighbors = get_similar(image,n_neighbors=11)

    plt.figure(figsize=[8,6])
    plt.subplot(3,4,1)
    plt.imshow(image.cpu().numpy().transpose([1,2,0]))
    plt.title("Original image")

    for i in range(11):
        plt.subplot(3,4,i+2)
        plt.imshow(neighbors[i].cpu().numpy().transpose([1,2,0]))
        plt.title("Dist=%.3f"%distances[i])
    plt.show()

<тут выведите самые похожие лица к какому-нибудь лицу из тестовой
части датасета>
```