



Deep Learning School

Школа глубокого обучения ФПМИ МФТИ

Домашнее задание. Весна 2021

Autoencoders

Часть 1. Vanilla Autoencoder (10 баллов)

1.1. Подготовка данных (0.5 балла)

```
import numpy as np
import pandas as pd
import os
from torch.autograd import Variable
from torchvision import datasets
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
from sklearn.model_selection import train_test_split
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
%matplotlib inline

from skimage.transform import resize
import skimage.io
from IPython.display import clear_output
from sklearn.manifold import TSNE
import seaborn as sns

from torch.autograd import Variable
import torchvision
import copy
```

```
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, downloading...")
        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_path':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)\n                .apply(lambda img:img[dy:-dy,dx:-dx])\n                .apply(lambda img: resize(img,[dimx,dimy]))\n\nall_photos = np.stack(all_photos.values)
all_attrs = df\n\nreturn 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  
# from get_dataset import fetch_dataset  
all_photos, allAttrs = fetch_dataset()
```

```
images not found, downloading...
extracting...
done
attributes not found, downloading...
done
```

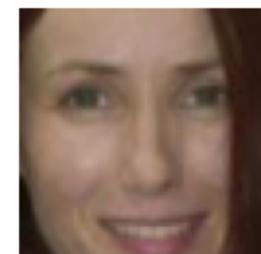
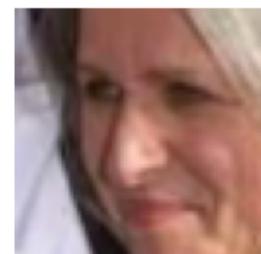
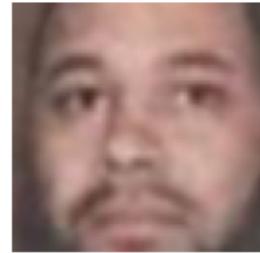
Разбейте выборку картинок на train и val, выведите несколько картинок в output, чтобы посмотреть, как они выглядят, и приведите картинки к тензорам pytorch, чтобы можно было скормить их сети:

```
train_photos, val_photos, train_attrs, val_attrs = train_test_split(all_photos, all_attrs,
                                                               train_size=0.9, shuffle=False)
train_loader = torch.utils.data.DataLoader(train_photos, batch_size=32)
val_loader = torch.utils.data.DataLoader(val_photos, batch_size=32)

device = torch.device("cuda")

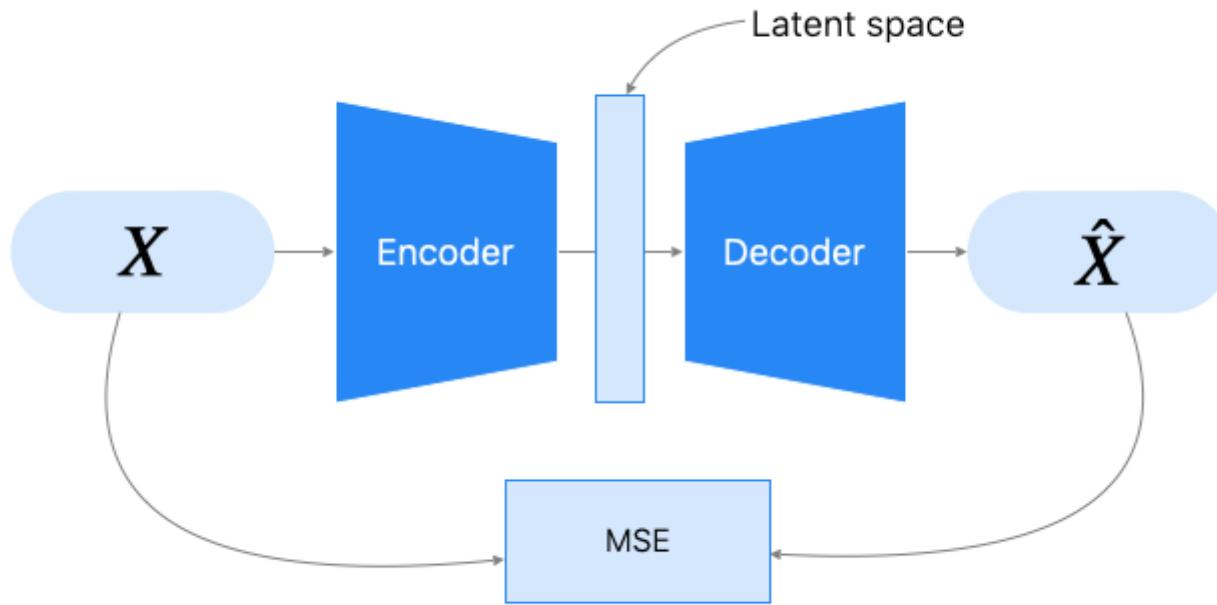
plt.figure(figsize=(18, 6))
for i in range(6):
    plt.subplot(2, 6, i+1)
    plt.axis("off")
    plt.imshow(train_photos[i])

    plt.subplot(2, 6, i+7)
    plt.axis("off")
    plt.imshow(val_photos[i])
plt.show();
```



1.2. Архитектура модели (1.5 балла)

В этом разделе мы напишем и обучем обычный автоэнкодер.



^ напомню, что автоэнкодер выглядит вот так

```
# выберите размер латентного вектора
LATENT_DIM = 100
```

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

```
features = LATENT_DIM

# define a simple linear VAE
class LinearVAE(nn.Module):
    def __init__(self):
        super(LinearVAE, self).__init__()
```

```
"""
source: seminar
"""

self.flatten = nn.Flatten()

# encoder
self.encoder = nn.Sequential(
    nn.Linear(in_features=12288, out_features=512),
    nn.ReLU(),
    nn.Linear(in_features=512, out_features=features*2)

)

# decoder
self.decoder = nn.Sequential(
    nn.Linear(in_features=features, out_features=512),
    nn.ReLU(),
    nn.Linear(in_features=512, out_features=12288)
)

def reparameterize(self, mu, log_var):
    """
    :param mu: mean from the encoder's latent space
    :param log_var: log variance from the encoder's latent space
    """
    std = torch.exp(0.5 * log_var) # standard deviation
    eps = torch.randn_like(std) # `randn_like` as we need the same size
    sample = mu + (eps * std) # sampling as if coming from the input space
    return sample

def forward(self, x):
    # encoding
    x = self.flatten(x).float()
    x = self.encoder(x).view(-1, 2, features)
    # get `mu` and `log_var`
    mu = x[:, 0, :] # the first feature values as mean
```

```
log_var = x[:, 1, :] # the other feature values as variance
# get the latent vector through reparameterization
z = self.reparameterize(mu, log_var)

# decoding
x = self.decoder(z)
reconstruction = torch.sigmoid(x)
return reconstruction, mu, log_var

def sample(self, z):
    generated = self.decoder(z)
    generated = torch.sigmoid(generated)
    generated = generated.view(-1, 64, 64, 3)
    return generated

def get_latent_vector(self, x):
    x = self.flatten(x).float()
    x = self.encoder(x).view(-1, 2, features)
    # get `mu` and `log_var`
    mu = x[:, 0, :] # the first feature values as mean
    log_var = x[:, 1, :] # the other feature values as variance
    # get the latent vector through reparameterization
    z = self.reparameterize(mu, log_var)
    return z

criterion = nn.MSELoss()

model = LinearVAE().to(device)

optimizer = optim.Adam(model.parameters(), lr=1e-4)
```

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. Например, после каждой эпохи (прогона train выборки через автоэнкодер) 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 =)

```
n_epochs = 50
train_losses = []
val_losses = []

for epoch in tqdm(range(n_epochs)):
    model.train()
    train_losses_per_epoch = []
    for batch in train_loader:
        optimizer.zero_grad()
        reconstruction, mu, logsigma = model(batch.to(device))
        reconstruction = reconstruction.view(-1, 64, 64, 3)
        loss = criterion(batch.to(device).float(), reconstruction)
        loss.backward()
        optimizer.step()
        train_losses_per_epoch.append(loss.item())

    avg_loss = np.mean(train_losses_per_epoch)
    train_losses.append(avg_loss)

    clear_output(wait=True)
    row_names = ['Image', 'Reconstruction']
    X_val = next(iter(val_loader))
    X_val_reconstructed, _, _ = model(X_val.to(device))

    fig, big_axes = plt.subplots( figsize=(12, 8) , nrows=2, ncols=1, sharey=True)
    for row, big_ax in enumerate(big_axes, start=1):
        big_ax.set_title(row_names[row-1], fontsize=16)
        big_ax._frameon = False
```

```
for k in range(6):
    ax = fig.add_subplot(3, 6, k+1)
    ax.imshow(X_val[k].detach().cpu().numpy(), cmap='gray')
    plt.axis('off')
    fig.suptitle('%d / %d - loss: %f' % (epoch+1, n_epochs, avg_loss))

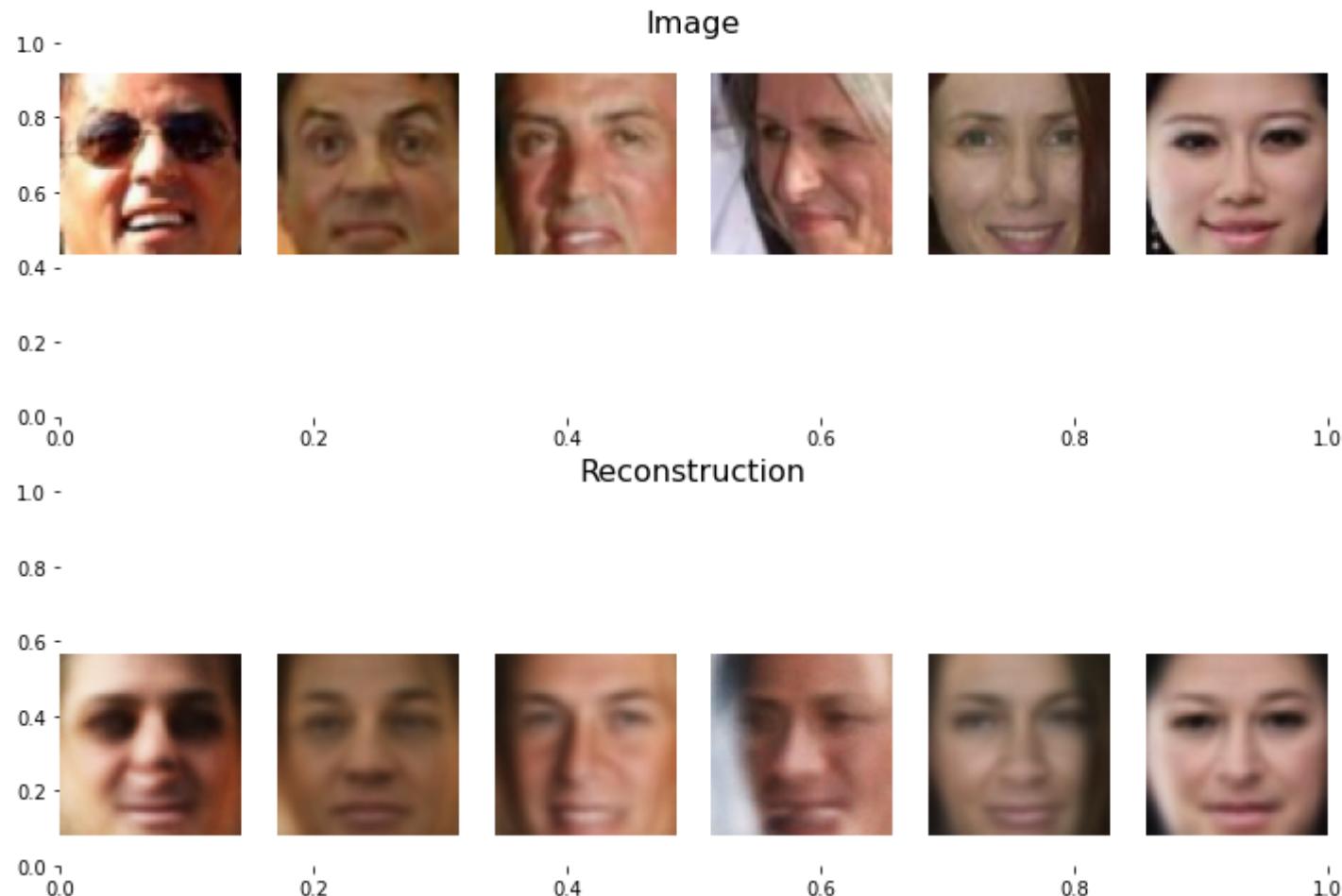
    ax = fig.add_subplot(3, 6, k+13)
    plt.imshow(X_val_reconstructed[k].view(X_val[k].shape).detach().cpu().numpy(), cmap='gray')
    plt.axis('off')

fig.set_facecolor('w')
plt.tight_layout()
plt.show()

model.eval()
val_losses_per_epoch = []
with torch.no_grad():
    for batch in val_loader:
        reconstruction, mu, logsigma = model(batch.to(device))
        reconstruction = reconstruction.view(-1, 64, 64, 3)
        loss = criterion(batch.to(device).float(), reconstruction)
        val_losses_per_epoch.append(loss.item())

val_losses.append(np.mean(val_losses_per_epoch))
```

50 / 50 - loss: 0.003584



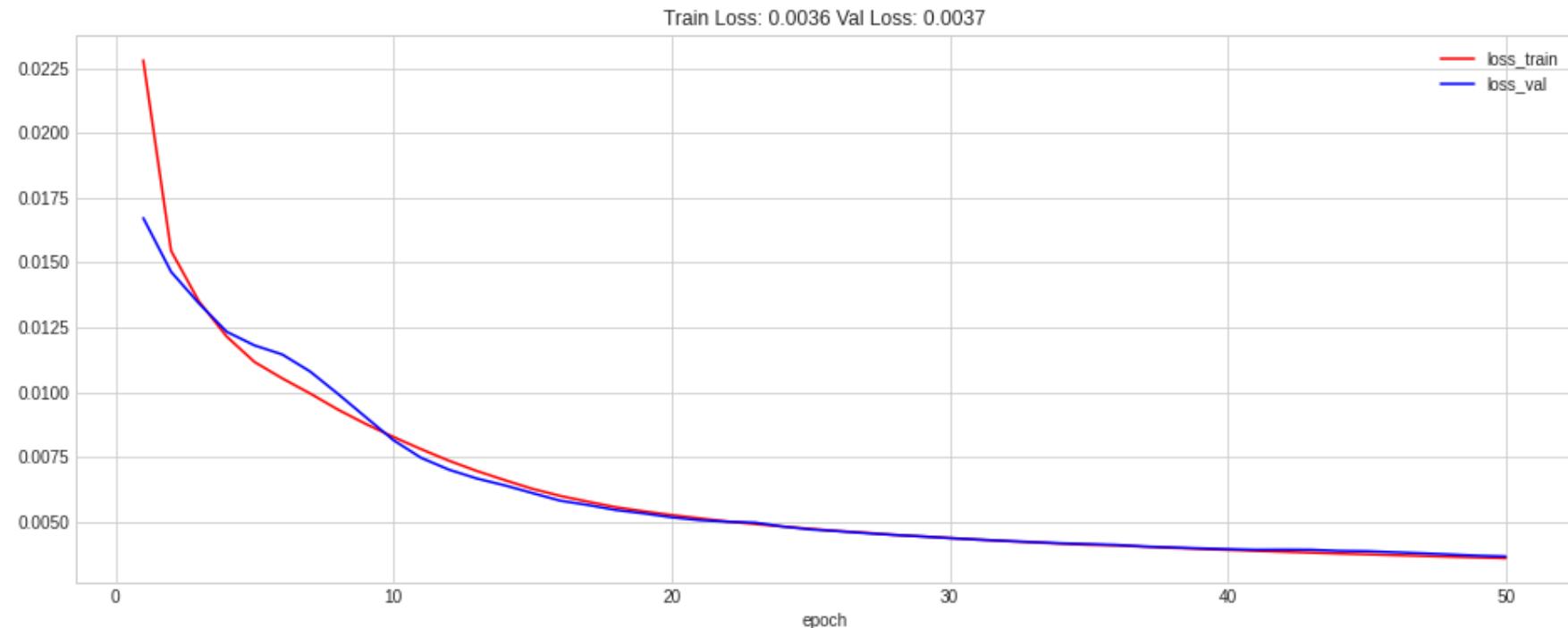
```
def plot_learning_curves(train_losses, val_losses, title=""):  
    metric_history = {"train": [{"epoch": i, "loss": v} for i, v in enumerate(train_losses, 1)],  
                      "val": [{"epoch": i, "loss": v} for i, v in enumerate(val_losses, 1)]}  
    with plt.style.context('seaborn-whitegrid'):  
        fig, ax = plt.subplots(1, 1, figsize=(16, 6))  
        train_history = pd.DataFrame(metric_history["train"])
```

```

val_history = pd.DataFrame(metric_history["val"])
train_history.plot(x="epoch", y="loss", color="r", ax=ax, label="loss_train")
val_history.plot(x="epoch", y="loss", color="b", ax=ax, label="loss_val")
ax.set_title(f'Train Loss: {train_history.iloc[-1]["loss"]:.4f} Val Loss: {val_history.iloc[-1]["loss"]:.4f}')
if not title:
    fig.suptitle(title)
plt.show();

```

```
plot_learning_curves(train_losses, val_losses)
```



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

```

X_val = next(iter(val_loader))
X_val_reconstructed, _, _ = model(X_val.to(device))

```

```
plt.figure(figsize=(18, 6))
for i in range(6):
    plt.subplot(2, 6, i+1)
    plt.axis("off")
    plt.imshow(X_val[i].detach().cpu().numpy())

    plt.subplot(2, 6, i+7)
    plt.axis("off")
    plt.imshow(X_val_reconstructed[i].view(X_val[k].shape).detach().cpu().numpy())

plt.show();
```



Not bad, right?

1.4. Sampling (2 балла)

Давайте теперь будем не просто брать картинку, прогонять ее через автоэнкодер 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: Если 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.

```
# сгенерируем 25 рандомных векторов размера latent_space
z = np.random.randn(25, LATENT_DIM)
output = model.sample(torch.FloatTensor(z).to(device))

plt.figure(figsize=(18, 6))
for i in range(6):
    plt.subplot(2, 6, i+1)
    plt.axis("off")
    plt.imshow(output[i].detach().cpu().numpy())

plt.show();
```



Time to make fun! (4 балла)

Давайте научимся пририсовывать людям улыбки =)

so linear

this is you when looking at the HW for the first time



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 присутствующие атрибуты (улыбки, очки...)

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

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

4. 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!

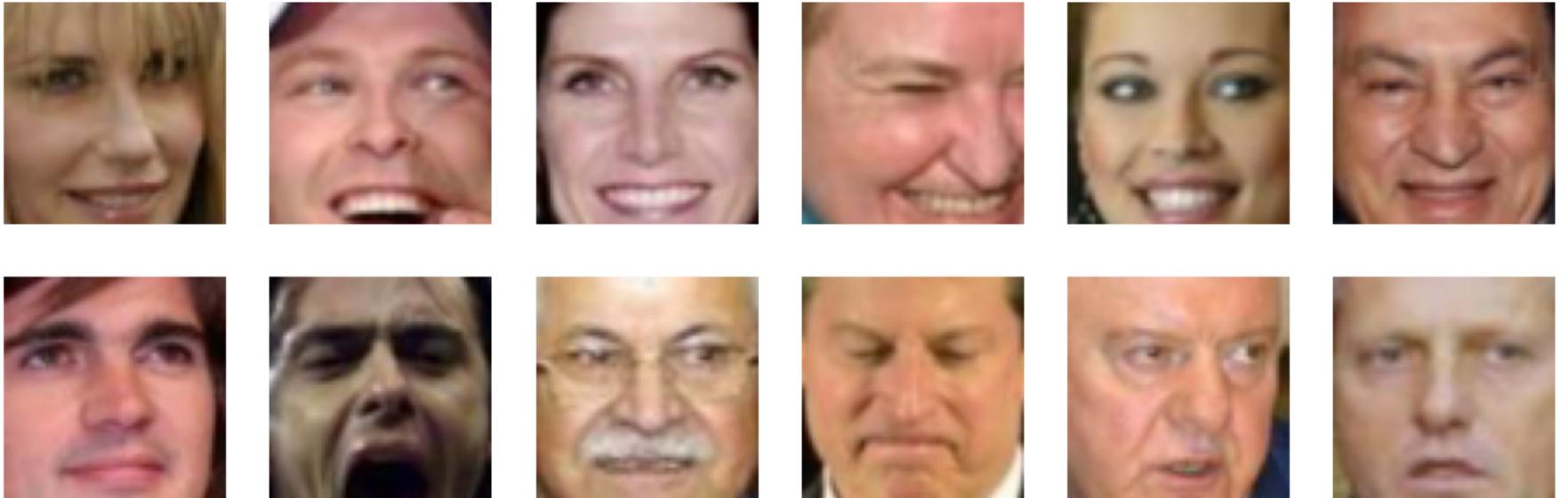
```
SAMPLE_SIZE = 100 # я решил увеличить вместо 15, хотя и на 15 работает также
```

```
smiling_people = all_attrs[all_attrs["Smiling"]>0.7].sample(SAMPLE_SIZE)
smiling_people_photos = all_photos[smiling_people.index]
```

```
non_smiling_people = all_attrs[all_attrs["Smiling"]<0.2].sample(SAMPLE_SIZE)
non_smiling_people_photos = all_photos[non_smiling_people.index]
```

```
plt.figure(figsize=(18, 6))
for i in range(6):
    plt.subplot(2, 6, i+1)
    plt.axis("off")
    plt.imshow(smiling_people_photos[i])

    plt.subplot(2, 6, i+7)
    plt.axis("off")
    plt.imshow(non_smiling_people_photos[i])
plt.show();
```



```
smiling_people_loader = torch.utils.data.DataLoader(smiling_people_photos, batch_size=len(smiling_people_photos))
non_smiling_people_loader = torch.utils.data.DataLoader(non_smiling_people_photos, batch_size=len(non_smiling_people
```

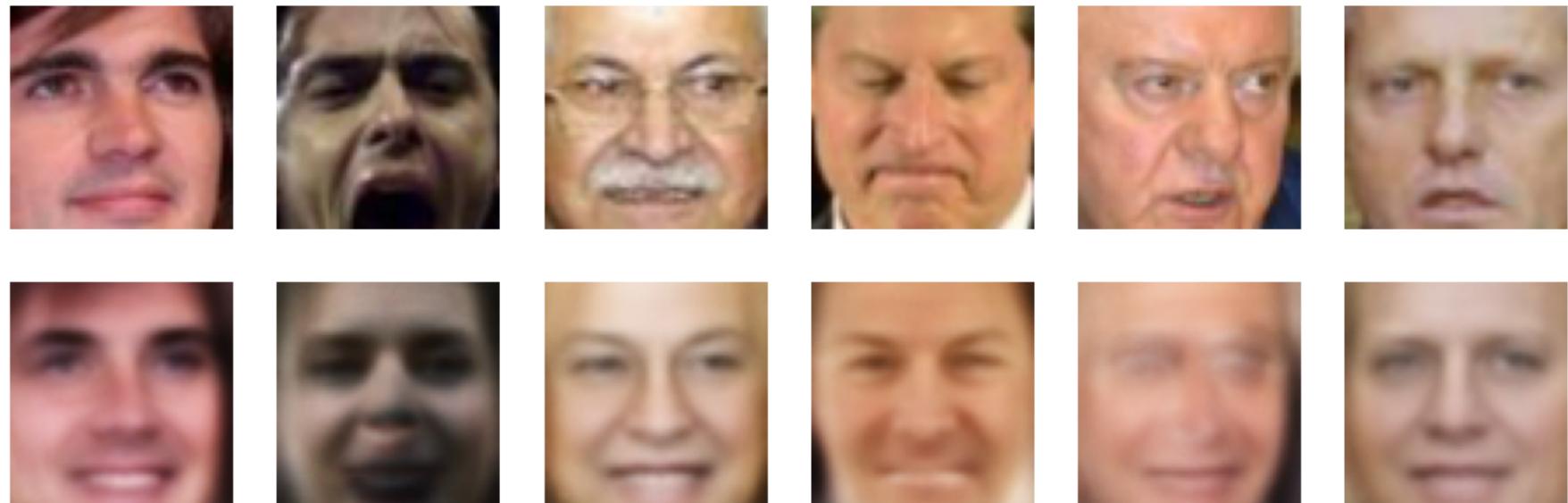
```
smiling_people_latent_vector = model.get_latent_vector(next(iter(smiling_people_loader)).to(device))
non_smiling_people_latent_vector = model.get_latent_vector(next(iter(non_smiling_people_loader)).to(device))
```

```
smile_vector = smiling_people_latent_vector.mean(0)-non_smiling_people_latent_vector.mean(0)
```

```
z = non_smiling_people_latent_vector+smile_vector
x = model.decoder(z)
reconstruction = torch.sigmoid(x)
```

```
plt.figure(figsize=(18, 6))
for i in range(6):
    plt.subplot(2, 6, i+1)
    plt.axis("off")
    plt.imshow(non_smiling_people_photos[i])

    plt.subplot(2, 6, i+7)
    plt.axis("off")
    plt.imshow(reconstruction[i].view(non_smiling_people_photos[i].shape).detach().cpu().numpy())
plt.show();
```



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 = 32
# MNIST Dataset
train_dataset = datasets.MNIST(root='./mnist_data/', train=True, transform=torchvision.transforms.ToTensor(), download=False)
test_dataset = datasets.MNIST(root='./mnist_data/', train=False, transform=torchvision.transforms.ToTensor(), download=False)

# Data Loader (Input Pipeline)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size, 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-
```

```
HBox(children=(FloatProgress(value=0.0, max=9912422.0), HTML(value='')))
```

```
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-
```

```
HBox(children=(FloatProgress(value=0.0, max=28881.0), HTML(value='')))
```

```
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-ub
```

```
HBox(children=(FloatProgress(value=0.0, max=1648877.0), HTML(value='')))
```

```
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-ub
```

```
HBox(children=(FloatProgress(value=0.0, max=4542.0), HTML(value='')))
```

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

```
Processing...
```

```
Done!
```

```
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:502: UserWarning: The given NumPy array is not  
    return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
```

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!

```
img_height = 28
img_width = 28
n_channels = 1

class Encoder(nn.Module):
    def __init__(self, dim=LATENT_DIM, act=F.relu):
        super(Encoder, self).__init__()

        self.linear1 = nn.Linear(img_height * img_width * n_channels, 512)
        self.linear2 = nn.Linear(512, 256)
        self.linear3 = nn.Linear(256, 200)

        self.mu = nn.Linear(200, dim)
        self.var = nn.Linear(200, dim)

        self.act = act

    def forward(self, x):

        hidden1 = self.act(self.linear1(x))
        hidden2 = self.act(self.linear2(hidden1))
        hidden3 = self.act(self.linear3(hidden2))

        mu = self.mu(hidden3)
        logsigma = self.var(hidden3)
        return mu, logsigma


class Decoder(nn.Module):
    def __init__(self, dim=LATENT_DIM, act=F.relu):
        super(Decoder, self).__init__()
        self.latent_to_hidden = nn.Linear(dim, 200)
```

```
self.hidden_to_out1 = nn.Linear(200, 256)
self.hidden_to_out2 = nn.Linear(256, 512)
self.hidden_to_out3 = nn.Linear(512, img_height * img_width * n_channels)

self.act = act

def forward(self, z):

    x = self.act(self.latent_to_hidden(z))
    x = self.act(self.hidden_to_out1(x))
    x = self.act(self.hidden_to_out2(x))

    reconstruction = F.sigmoid(self.hidden_to_out3(x))
    return reconstruction

class VAE(nn.Module):
    def __init__(self, dim=LATENT_DIM, act=F.relu):
        super(VAE, self).__init__()
        # определите архитектуры encoder и decoder
        # помните, у encoder должны быть два "хвоста",
        # т.е. encoder должен кодировать картинку в 2 переменные -- mu и logsigma
        self.encoder = Encoder(dim=dim, act=act)
        self.decoder = Decoder(dim=dim, act=act)

    def encode(self, x):
        # реализуйте forward проход энкодера
        # в качестве возвращаемых переменных -- mu и logsigma
        mu, logsigma = self.encoder(x)
        return mu, logsigma

    def gaussian_sampler(self, mu, logsigma):
        if self.training:
            # засемплируйте латентный вектор из нормального распределения с параметрами mu и sigma
            std = logsigma.exp_()
            eps = Variable(std.data.new(std.size()).normal_())
            return eps.mul(std).add_(mu)
        else:
```

```

# на инференсе возвращаем не случайный вектор из нормального распределения, а центральный -- ти.
# на инференсе выход автоэнкодера должен быть детерминирован.
return mu

def decode(self, z):
    # реализуйте forward проход декодера
    # в качестве возвращаемой переменной -- reconstruction
    reconstruction = self.decoder(z)
    return reconstruction

def forward(self, x):
    # используя encode и decode, реализуйте forward проход автоэнкодера
    # в качестве возвращаемых переменных -- mu, logsigma и reconstruction
    mu, logsigma = self.encoder(x)
    latent = self.gaussian_sampler(mu, logsigma)
    reconstruction = self.decoder(latent)
    return reconstruction, mu, logsigma

def get_latent_vector(self, x):
    mu, logsigma = self.encoder(x)
    latent = self.gaussian_sampler(mu, logsigma)
    return latent

```

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:

$$\mathcal{L} = -D_{KL}(q_\phi(z|x)||p(z)) + \log p_\theta(x|z)$$

Формула для KL-дивергенции:

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

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

```
def KL_divergence(mu, logsigma):
    """
    часть функции потерь, которая отвечает за "близость" латентных представлений разных людей
    """
    loss = -0.5 * torch.sum(1 + logsigma - mu.pow(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)) / x.size(0)

def loss_vae_beta(x, mu, logsigma, reconstruction, beta=0.2):
    return beta*KL_divergence(mu, logsigma) / x.size(0) + (1-beta)*log_likelihood(x, reconstruction)
```

И обучим модель:

```
criterion = loss_vae_beta

model = VAE().to(device)

optimizer = optim.Adam(model.parameters(), lr=1e-4)
```

```
n_epochs = 50
train_losses = []
val_losses = []
best_loss = np.inf
best_model_wts = None

for epoch in tqdm(range(n_epochs)):
    model.train()
    train_losses_per_epoch = []
    for batch, _ in train_loader:
        optimizer.zero_grad()
        reconstruction, mu, logsigma = model(torch.flatten(batch.to(device), start_dim=1))
        reconstruction = reconstruction.view(-1, n_channels, img_height, img_width)
        loss = criterion(batch.to(device).float(), reconstruction=reconstruction, mu=mu, logsigma=logsigma)
        loss.backward()
        optimizer.step()
        train_losses_per_epoch.append(loss.item())

    avg_loss = np.mean(train_losses_per_epoch)
    train_losses.append(avg_loss)

    model.eval()
    val_losses_per_epoch = []
    with torch.no_grad():
        for batch, _ in test_loader:
            reconstruction, mu, logsigma = model(torch.flatten(batch.to(device), start_dim=1))
            reconstruction = reconstruction.view(-1, n_channels, img_height, img_width)
            loss = criterion(batch.to(device).float(), reconstruction=reconstruction, mu=mu, logsigma=logsigma)
            val_losses_per_epoch.append(loss.item())

    val_loss = np.mean(val_losses_per_epoch)
    val_losses.append(val_loss)

    if best_loss>val_loss:
        best_loss = val_loss
        best_model_wts = copy.deepcopy(model.state_dict())
```

```
clear_output(wait=True)
row_names = ['Image', 'Reconstruction']
X_val = next((batch for batch, _ in test_loader))
X_val_reconstructed, _, _ = model(torch.flatten(X_val.to(device), start_dim=1))

fig, big_axes = plt.subplots( figsize=(12, 8) , nrows=2, ncols=1, sharey=True)
for row, big_ax in enumerate(big_axes, start=1):
    big_ax.set_title(row_names[row-1], fontsize=16)
    big_ax._frameon = False

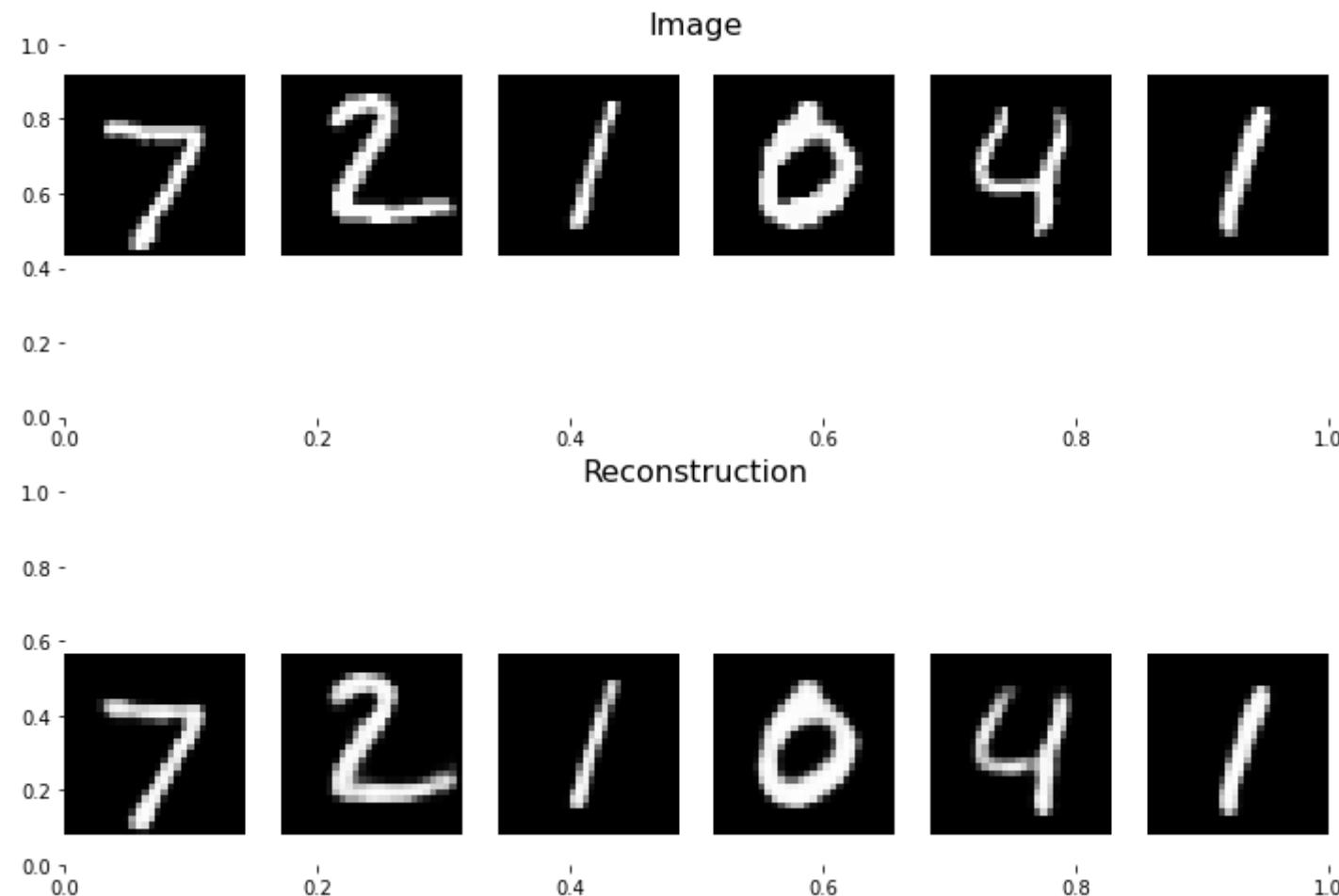
for k in range(6):
    ax = fig.add_subplot(3, 6, k+1)
    img = np.stack((X_val[k].permute(1,2,0).squeeze(-1).detach().cpu().numpy(),)*3, axis=-1)
    ax.imshow(img, cmap='gray')
    plt.axis('off')
    fig.suptitle('%d / %d - loss: %f' % (epoch+1, n_epochs, val_loss))

    ax = fig.add_subplot(3, 6, k+13)
    img = np.stack((X_val_reconstructed[k].view(X_val[k].shape).permute(1,2,0).squeeze(-1).detach().cpu().numpy(),
    plt.imshow(img, cmap='gray')
    plt.axis('off')

fig.set_facecolor('w')
plt.tight_layout()
plt.show()

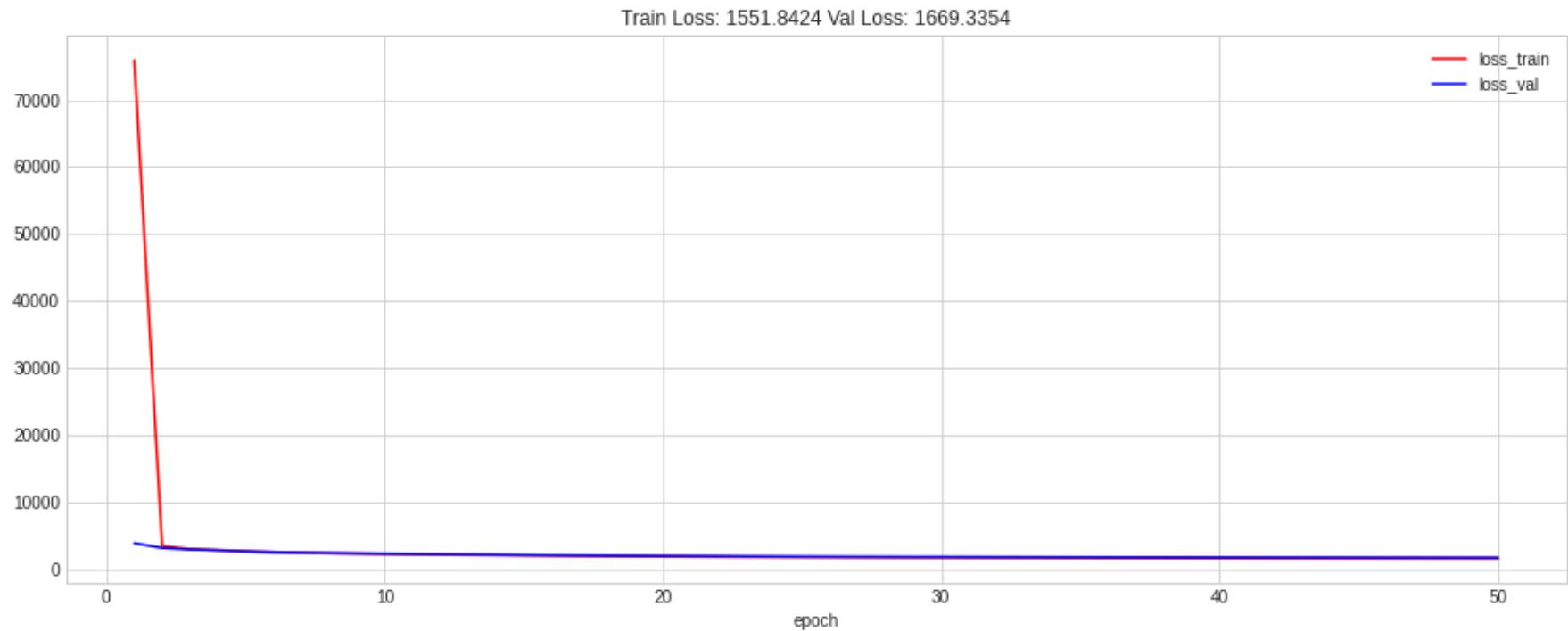
model.load_state_dict(best_model_wts)
```

50 / 50 - loss: 1669.335418



<All keys matched successfully>

```
plot_learning_curves(train_losses, val_losses)
```



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

```
X_val = next((batch for batch, _ in test_loader))
X_val_reconstructed, _, _ = model(torch.flatten(X_val.to(device), start_dim=1))

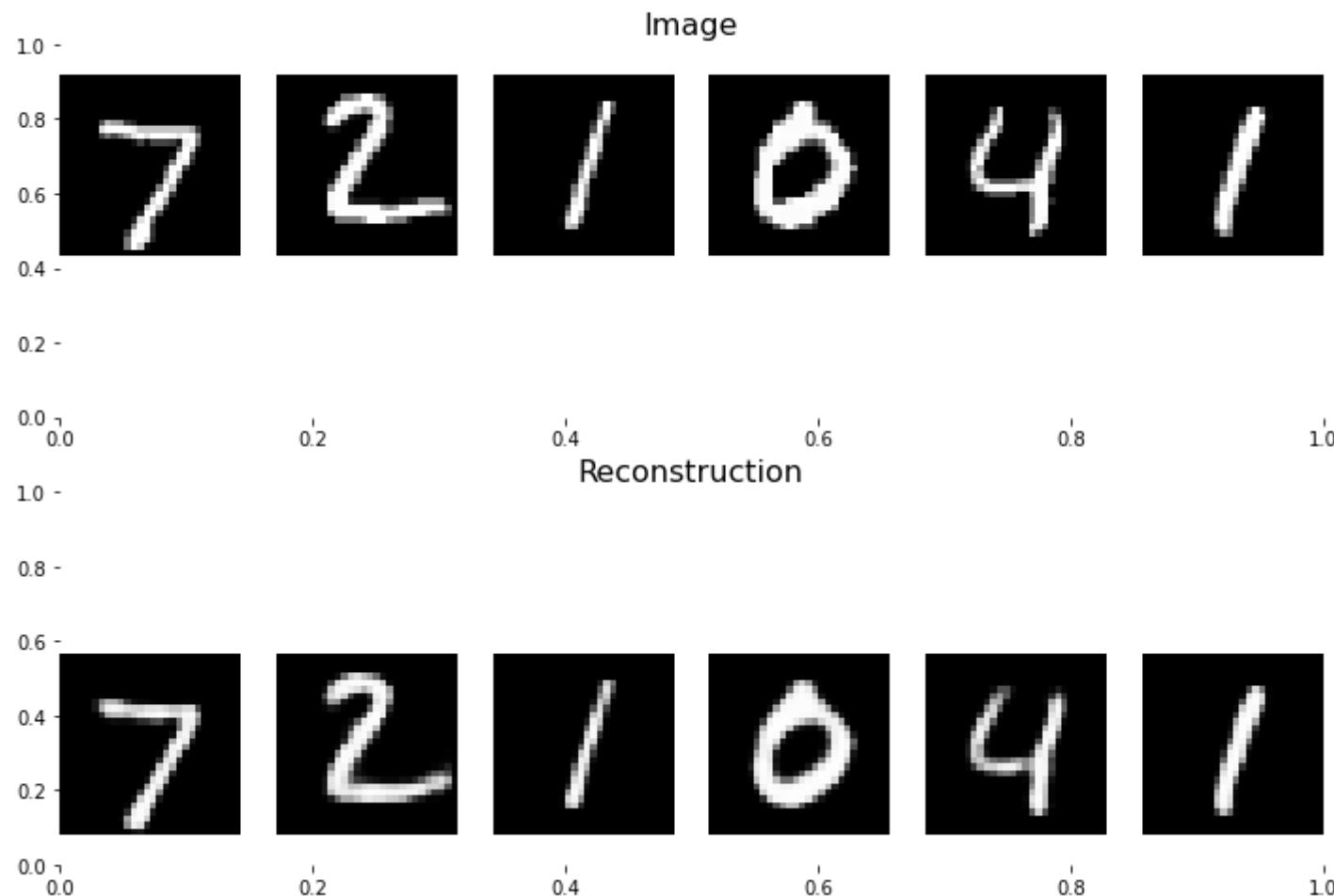
fig, big_axes = plt.subplots( figsize=(12, 8) , nrows=2, ncols=1, sharey=True)
for row, big_ax in enumerate(big_axes, start=1):
    big_ax.set_title(row_names[row-1], fontsize=16)
    big_ax._frameon = False

for k in range(6):
    ax = fig.add_subplot(3, 6, k+1)
    img = np.stack((X_val[k].permute(1,2,0).squeeze(-1).detach().cpu().numpy(),)*3, axis=-1)
    ax.imshow(img, cmap='gray')
    plt.axis('off')
    fig.suptitle('%d / %d - loss: %f' % (epoch+1, n_epochs, avg_loss))
```

```
ax = fig.add_subplot(3, 6, k+13)
img = np.stack((X_val_reconstructed[k].view(X_val[k].shape).permute(1,2,0).squeeze(-1).detach().cpu().numpy(),)*
plt.imshow(img, cmap='gray')
plt.axis('off')
```

/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1709: UserWarning: nn.functional.sigmoid is deprecated
warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")

50 / 50 - loss: 1551.842418

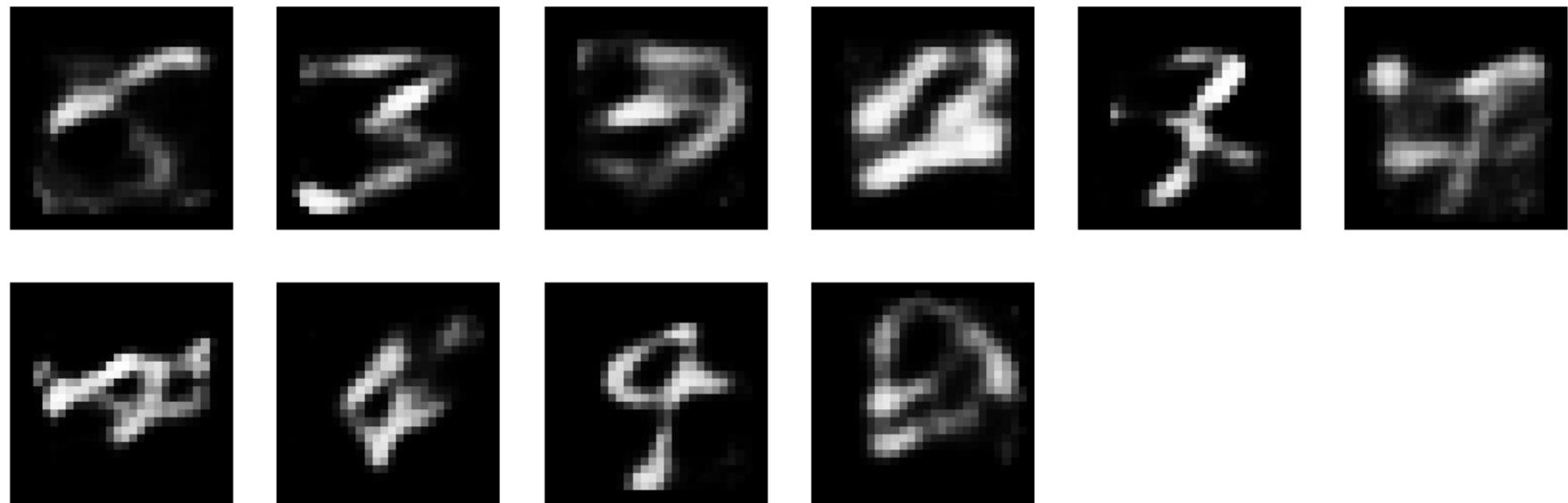


Давайте попробуем проделать для VAE то же, что и с обычным автоэнкодером -- подсунуть decoder'у из VAE случайные векторы из нормального распределения и посмотреть, какие картинки получаются:

```
# вспомните про замечание из этого же пункта обычного AE про распределение латентных переменных
z = np.array([np.random.normal(0, 1, 100) for i in range(10)])
z = torch.FloatTensor(z).to(device)
output = model.decoder(z)
# <выведите тут полученные картинки>
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1709: UserWarning: nn.functional.sigmoid is deprecated  
warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
```

```
plt.figure(figsize=(18, 6))  
for i, vector in enumerate(output):  
    plt.subplot(2, 6, i+1)  
    plt.axis("off")  
    img = np.stack((vector.view(-1, img_height, img_width).permute(1,2,0).squeeze(-1).detach().cpu().numpy(),)*3, axis=0)  
    plt.imshow(img)  
  
plt.show();
```



хм...цифры, восстановленные из случайных векторов подозрительно похожи на цифры из фильма Хищник;) видимо действительно нужно сэмплировать в соответствии с распределением, которое у нас есть. кстати еще по логике

вещей можно reparametrization улучшить используя gumbel softmax <https://arxiv.org/pdf/1611.01144.pdf>

с другой стороны мы можем как в семинаре взять ground truth и получив latent vector , сгенерировать изображения

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 пространстве.

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 помощью 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.

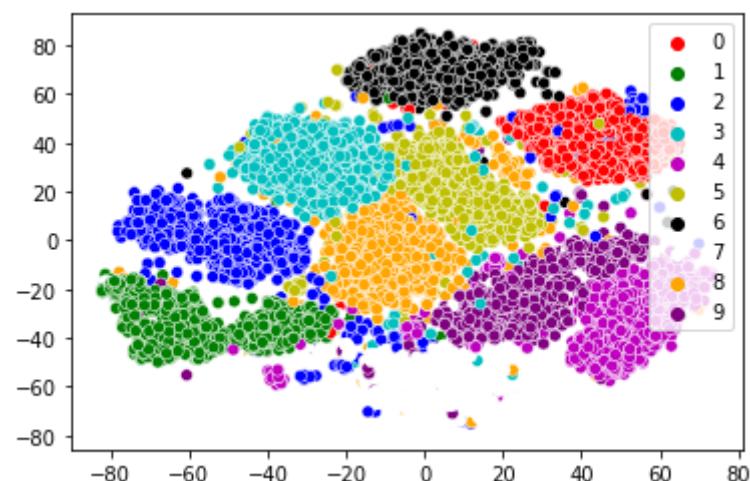
```
labels = []
vectors = []
for batch, l in test_loader:
    latent_vector = model.get_latent_vector(torch.flatten(batch.to(device), start_dim=1)).detach().cpu().numpy()
    vectors.extend(latent_vector)
    labels.extend(l.tolist())
vectors = np.array(vectors)
```

```
X_2d = TSNE(n_components=2).fit_transform(vectors)
```

```
colors = {0: 'r',
1: 'g',
2: 'b',
3: 'c',
4: 'm',
5: 'y',
6: 'k',
7: 'w',
8: 'orange',
9: 'purple'}
```

```
sns.scatterplot(x=X_2d[:, 0], y=X_2d[:, 1], hue=labels, palette=colors)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f48f83add10>
```



Что вы думаете о виде латентного представления?

в общем логично, что латентные представления достаточно неплохо отображают наши данные, в этом их и предназначение

Congrats v2.0!

2.3. Conditional VAE (6 баллов)

Мы уже научились обучать обычный АЕ на датасете картинок и получать новые картинки, используя генерацию шума и декодер. Давайте теперь допустим, что мы обучили АЕ 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 АЕ функцию "выдай мне рандомное число из вот этого вот класса", где классов десять (цифры от 0 до 9 образуют десять классов). Conditional АЕ — так называется вид автоэнкодера, который предоставляет такую возможность. Ну, название "conditional" уже говорит само за себя.

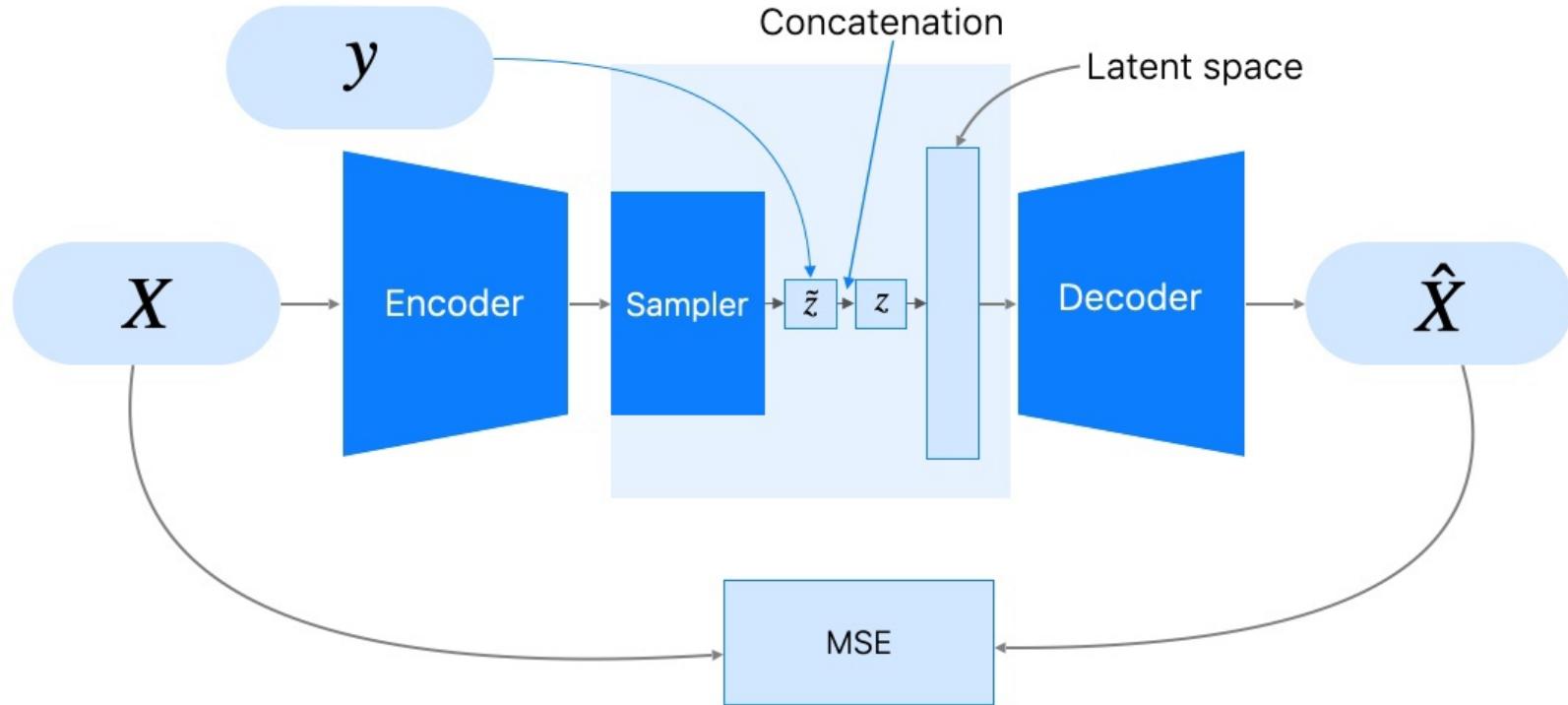
И в этой части задания мы научимся такие обучать.

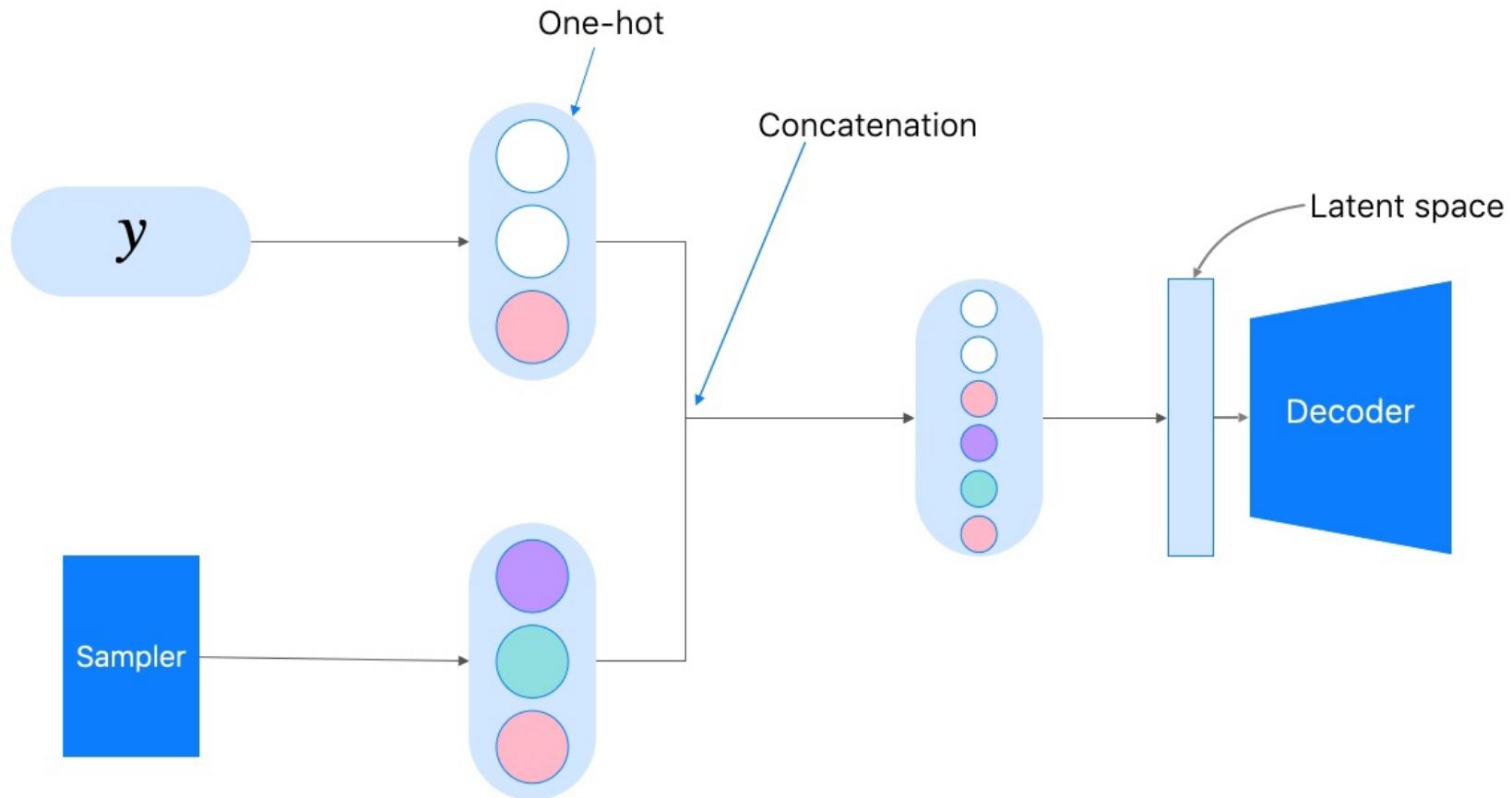
Архитектура

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 μ и σ

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 сконкатенировать выход из предыдущего слоя и информацию о классе.

```
class Encoder(nn.Module):
    def __init__(self, dim=LATENT_DIM, num_classes = 10, act=F.relu):
```

```
super(Encoder, self).__init__()
self.num_classes = num_classes

self.linear1 = nn.Linear(img_height * img_width * n_channels + num_classes, 512)
self.linear2 = nn.Linear(512, 256)
self.linear3 = nn.Linear(256, 200)

self.mu = nn.Linear(200, dim)
self.var = nn.Linear(200, dim)

self.act = act

def forward(self, x, class_num):
    if class_num is not None:
        class_num = idx2onehot(class_num, n=self.num_classes)
        x = torch.cat((x, class_num), dim=-1)

    hidden1 = self.act(self.linear1(x))
    hidden2 = self.act(self.linear2(hidden1))
    hidden3 = self.act(self.linear3(hidden2))

    mu = self.mu(hidden3)
    logsigma = self.var(hidden3)
    return mu, logsigma

def idx2onehot(idx, n):
    """
    https://discuss.pytorch.org/t/convert-int-into-one-hot-format/507/4
    """
    if idx.dim() == 1:
        idx = idx.unsqueeze(1)
    onehot = torch.zeros(idx.size(0), n).to(idx.device)
    onehot.scatter_(1, idx, 1)
    return onehot
```

```
class Decoder(nn.Module):
    def __init__(self, dim=LATENT_DIM, num_classes=10, act=F.relu):
        super(Decoder, self).__init__()
        self.num_classes = num_classes

        self.latent_to_hidden = nn.Linear(dim + num_classes, 200)
        self.hidden_to_out1 = nn.Linear(200, 256)
        self.hidden_to_out2 = nn.Linear(256, 512)
        self.hidden_to_out3 = nn.Linear(512, img_height * img_width * n_channels)
        self.act = act

    def forward(self, z, class_num):

        if class_num is not None:
            class_num = idx2onehot(class_num, n=self.num_classes)
            z = torch.cat((z, class_num), dim=-1)

        x = self.act(self.latent_to_hidden(z))
        x = self.act(self.hidden_to_out1(x))
        x = self.act(self.hidden_to_out2(x))

        reconstruction = F.sigmoid(self.hidden_to_out3(x))
        return reconstruction

class CVAE(nn.Module):
    def __init__(self, dim=LATENT_DIM, num_classes = 10, act=F.relu):
        super(CVAE, self).__init__()
        # определите архитектуры encoder и decoder
        # помните, у encoder должны быть два "хвоста",
        # т.е. encoder должен кодировать картинку в 2 переменные -- mu и logsigma
        self.num_classes = num_classes
        self.encoder = Encoder(dim=dim, num_classes = num_classes, act=act)
        self.decoder = Decoder(dim=dim, num_classes = num_classes, act=act)

    def encode(self, x, class_num):
```

```
# реализуйте forward проход энкодера
# в качестве возвращаемых переменных -- mu, logsigma и класс картинки
mu, logsigma = self.encoder(x, class_num)
return mu, logsigma, class_num

def gaussian_sampler(self, mu, logsigma):
    if self.training:
        # засемплируйте латентный вектор из нормального распределения с параметрами mu и sigma
        std = logsigma.exp_()
        eps = Variable(std.data.new(std.size()).normal_())
        return eps.mul(std).add_(mu)
    else:
        # на инференсе возвращаем не случайный вектор из нормального распределения, а центральный -- mu.
        # на инференсе выход автоэнкодера должен быть детерминирован.
        return mu

def decode(self, z, class_num):
    # реализуйте forward проход декодера
    # в качестве возвращаемой переменной -- reconstruction
    reconstruction = self.decoder(z, class_num)
    return reconstruction

def forward(self, x, class_num=None):
    # используя encode и decode, реализуйте forward проход автоэнкодера
    # в качестве возвращаемых переменных -- mu, logsigma и reconstruction
    mu, logsigma = self.encoder(x, class_num)
    latent = self.gaussian_sampler(mu, logsigma)
    reconstruction = self.decoder(latent, class_num)
    return mu, logsigma, reconstruction

def encode(self, x):
    # реализуйте forward проход энкодера
    # в качестве возвращаемых переменных -- mu и logsigma
    mu, logsigma = self.encoder(x)
    return mu, logsigma

def get_latent_vector(self, x, class_num):
```

```
        mu, logsigma = self.encoder(x, class_num)
        latent = self.gaussian_sampler(mu, logsigma)
        return latent

criterion = loss_vae_beta

model = CVAE().to(device)

optimizer = optim.Adam(model.parameters(), lr=1e-4)

n_epochs = 50
train_losses = []
val_losses = []
best_loss = np.inf
best_model_wts = None

for epoch in tqdm(range(n_epochs)):
    model.train()
    train_losses_per_epoch = []
    for batch, y in train_loader:
        optimizer.zero_grad()
        mu, logsigma, reconstruction = model(torch.flatten(batch.to(device), start_dim=1), y.to(device))
        reconstruction = reconstruction.view(-1, n_channels, img_height, img_width)
        loss = criterion(batch.to(device).float(), reconstruction=reconstruction, mu=mu, logsigma=logsigma)
        loss.backward()
        optimizer.step()
        train_losses_per_epoch.append(loss.item())

    avg_loss = np.mean(train_losses_per_epoch)
    train_losses.append(avg_loss)

    model.eval()
    val_losses_per_epoch = []
    with torch.no_grad():
```

```

for batch, y in test_loader:
    mu, logsigma, reconstruction = model(torch.flatten(batch.to(device), start_dim=1), y.to(device))
    reconstruction = reconstruction.view(-1, n_channels, img_height, img_width)
    loss = criterion(batch.to(device).float(), reconstruction=reconstruction, mu=mu, logsigma=logsigma)
    val_losses_per_epoch.append(loss.item())

val_loss = np.mean(val_losses_per_epoch)
val_losses.append(val_loss)

if best_loss>val_loss:
    best_loss = val_loss
    best_model_wts = copy.deepcopy(model.state_dict())

clear_output(wait=True)
row_names = ['Image', 'Reconstruction']
X_val, y = next(((batch, y) for batch, y in test_loader))
_, _, X_val_reconstructed = model(torch.flatten(X_val.to(device), start_dim=1), y.to(device))

fig, big_axes = plt.subplots( figsize=(12, 8) , nrows=2, ncols=1, sharey=True)
for row, big_ax in enumerate(big_axes, start=1):
    big_ax.set_title(row_names[row-1], fontsize=16)
    big_ax._frameon = False

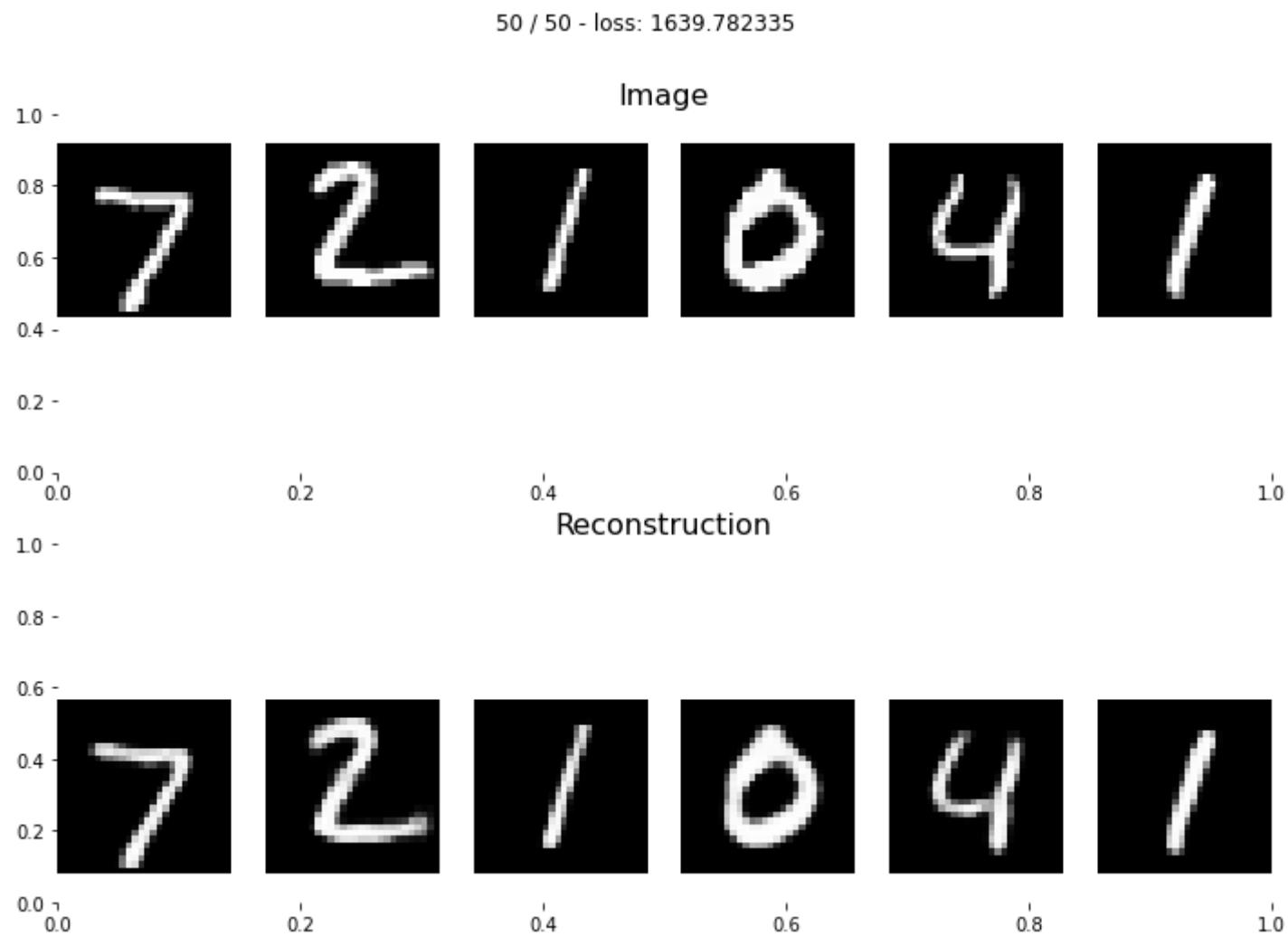
for k in range(6):
    ax = fig.add_subplot(3, 6, k+1)
    img = np.stack((X_val[k].permute(1,2,0).squeeze(-1).detach().cpu().numpy(),)*3, axis=-1)
    ax.imshow(img, cmap='gray')
    plt.axis('off')
    fig.suptitle('%d / %d - loss: %f' % (epoch+1, n_epochs, val_loss))

    ax = fig.add_subplot(3, 6, k+13)
    img = np.stack((X_val_reconstructed[k].view(X_val[k].shape).permute(1,2,0).squeeze(-1).detach().cpu().numpy(),
    plt.imshow(img, cmap='gray')
    plt.axis('off'))

fig.set_facecolor('w')
plt.tight_layout()

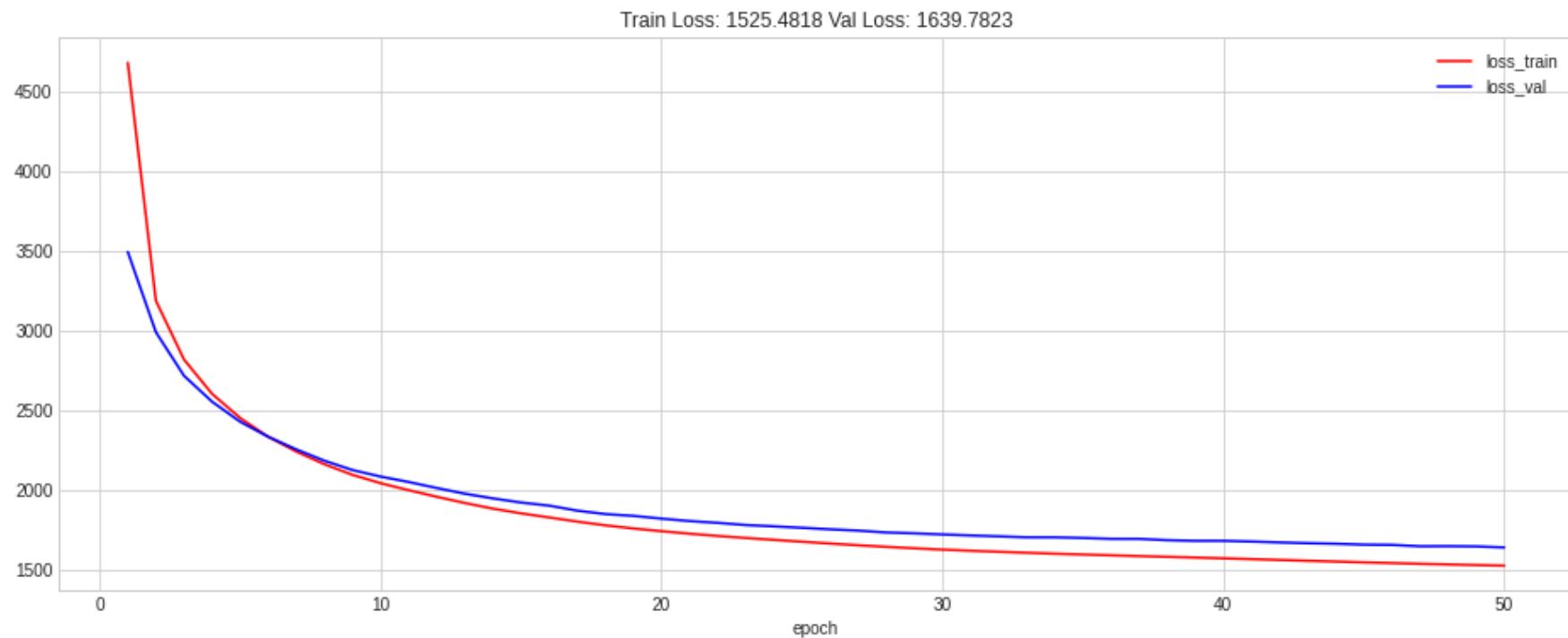
```

```
plt.show()  
  
model.load_state_dict(best_model_wts)
```



```
<All keys matched successfully>
```

```
plot_learning_curves(train_losses, val_losses)
```



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.

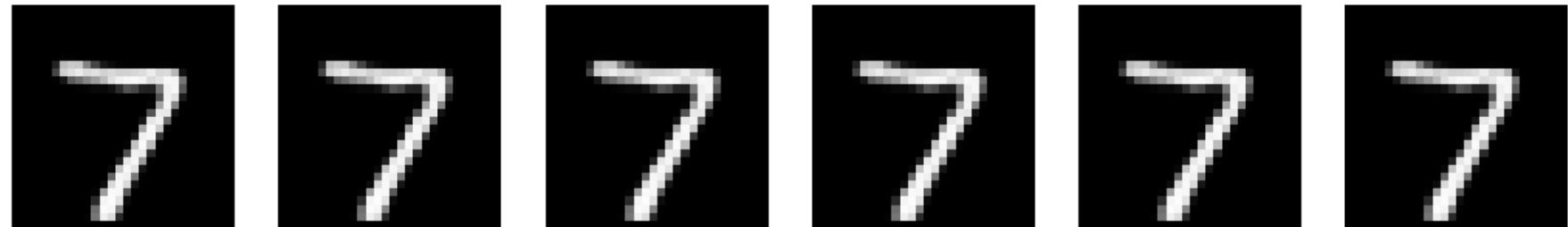
```
batch, y = next(iter(test_loader))
batch, y = torch.flatten(batch.to(device), start_dim=1), y.to(device)
```

```
latent_vector = model.get_latent_vector(batch, y).mean(0, keepdim=True)

plt.figure(figsize=(18, 6))
for i in range(6):
    plt.subplot(2, 6, i+1)
    plt.axis("off")
    z = torch.randn([1, LATENT_DIM]).to(device)
    vector = model.decode(latent_vector+z, torch.LongTensor([7]).to(device))
    img = np.stack((vector.view(-1, img_height, img_width).permute(1,2,0).squeeze(-1).detach().cpu().numpy(),)*3, axis=0)
    plt.imshow(img)

plt.show();
```

/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1709: UserWarning: nn.functional.sigmoid is deprecated
warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")



Splendid! Вы великолепны!

Latent Representations

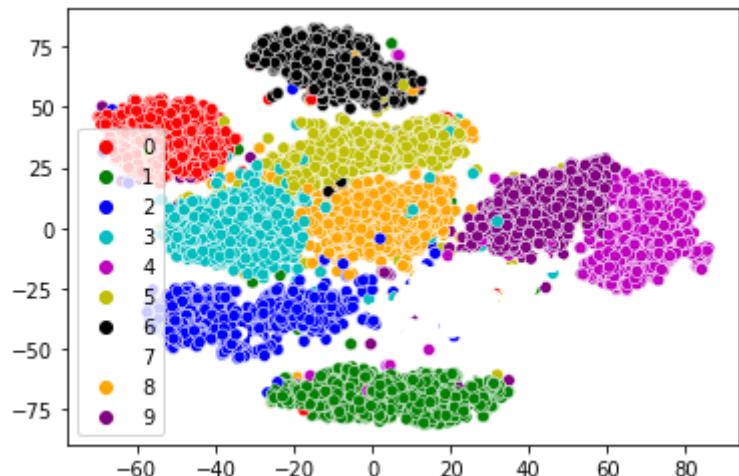
Давайте посмотрим, как выглядит латентное пространство картинок в CVAE и сравним с картинкой для VAE =)

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

```
# ваш код получения латентных представлений, применения TSNE и визуализации
labels = []
vectors = []
for batch, l in test_loader:
    latent_vector = model.get_latent_vector(torch.flatten(batch.to(device), start_dim=1), l.to(device)).detach().cpu()
    vectors.append(latent_vector)
    labels.append(l.tolist())
vectors = np.array(vectors)
```

```
X_2d = TSNE(n_components=2).fit_transform(vectors)
sns.scatterplot(x=X_2d[:, 0], y=X_2d[:, 1], hue=labels, palette=colors)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f490d4e0ad0>
```



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

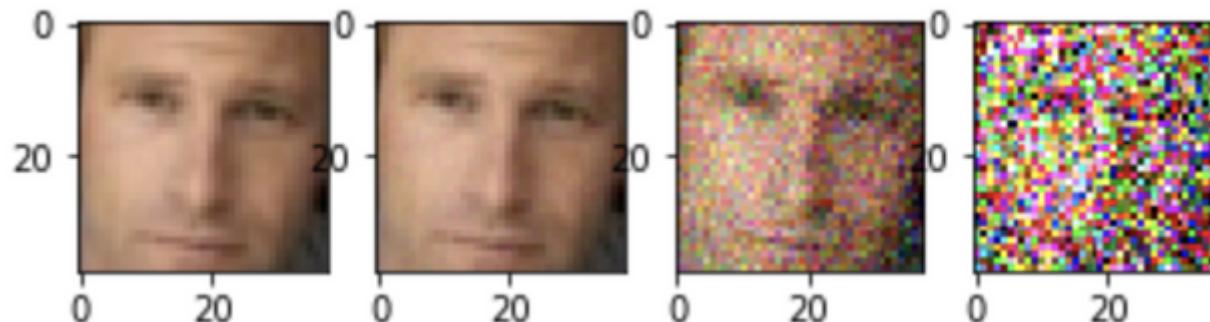
в целом они похожи, но как мы видим в случае CVAE существует больший margin между кластерами (похожие цифры, 4 и 9 например или 3 и 5, все также близки, что логично)

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 забудьте разбить на train/test!»

«тут проверка, как AE убирает шум 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 латентных представлений лиц нашей базы самые близайшие!

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 близайшие из латентных представлений тренировочной базы
5. Находим лица тренировочного датасета, которым соответствуют близайшие латентные представления, визуализируем!

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

```
codes = <получите 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»