

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 random
import os
import skimage.io
from IPython.display import clear_output
from skimage.transform import resize, rescale
```

```
from sklearn.model_selection import train_test_split
from torch.autograd import Variable
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
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 tqdm.notebook import tqdm
import matplotlib.pyplot as plt
%matplotlib inline

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)

cuda

random.seed(0)
np.random.seed(0)
torch.manual_seed(0)
torch.cuda.manual_seed(0)
torch.backends.cudnn.deterministic = True

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, downwloading...")
        os.system("wget http://vis-www.cs.umass.edu/lfw/lfw-deepfunneled.tgz -O tmp.t")
        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_p

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\n

all_photos = np.stack(all_photos.values).astype('float32')
all_attrs = df.drop(["photo_path","person","imagenum"],axis=1)

return all_photos, all_attrs

```

```

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:

```

batch_size = 32

train_photos, val_photos, train_attrs, val_attrs = train_test_split(data, attrs, tra
print("Training input shape: ", train_photos.shape)

data_tr = torch.utils.data.DataLoader(train_photos, batch_size=batch_size)
data_val = torch.utils.data.DataLoader(val_photos, batch_size=batch_size)

```

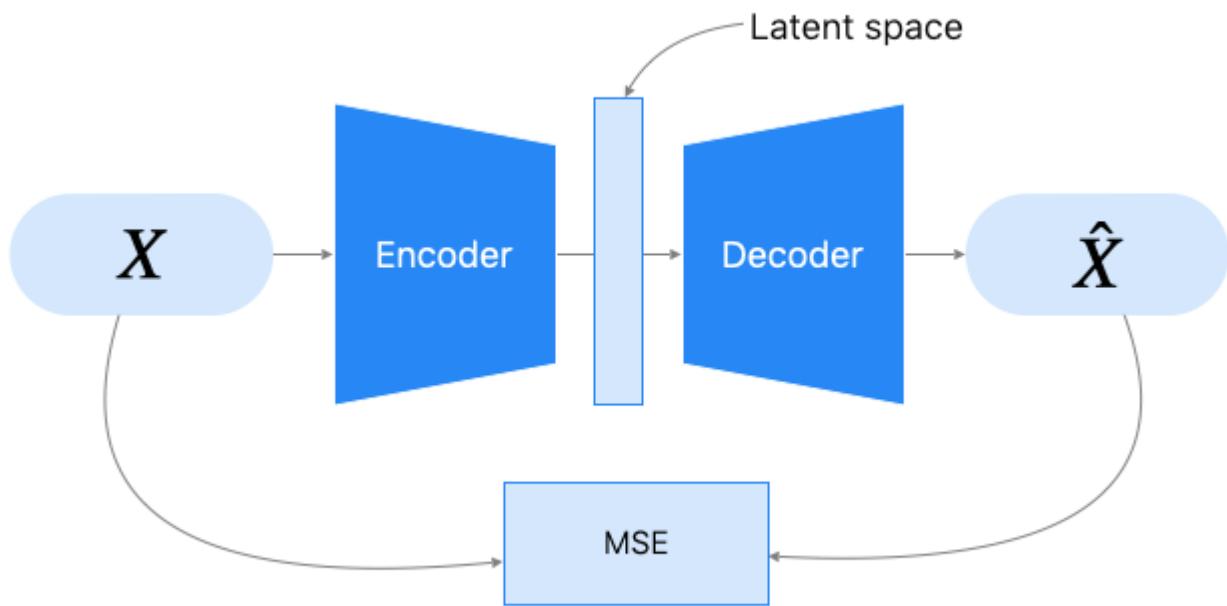
Training input shape: (11828, 64, 64, 3)

```
plt.figure(figsize=(18, 6))
for i in range(12):
    plt.subplot(2, 6, i+1)
    plt.axis("off")
    plt.imshow(data[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

```
dim_code = 64 # выберите размер латентного вектора
```

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

```
from copy import deepcopy

class Autoencoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.encoder = nn.Sequential(
            nn.Conv2d(64, 256, kernel_size=7, stride=1, padding=3),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 128, kernel_size=7, stride=1, padding=3),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.Conv2d(128, 16, kernel_size=7, stride=1, padding=3),
            nn.BatchNorm2d(16),
        )

        self.fc = nn.Sequential(
            nn.Linear(16 * 64 * 3, dim_code)
```

```

        )
self.unfc = nn.Sequential(
    nn.Linear(dim_code, 16 * 64 * 3),
)
self.decoder = nn.Sequential(
    nn.ConvTranspose2d(16, 128, kernel_size=7, stride=1, padding=3),
    nn.BatchNorm2d(128),
    nn.ReLU(),
    nn.ConvTranspose2d(128, 256, kernel_size=7, stride=1, padding=3),
    nn.BatchNorm2d(256),
    nn.ReLU(),
    nn.ConvTranspose2d(256, 64, kernel_size=7, stride=1, padding=3),
)

```

```

def forward(self, x):
    x = self.encoder(x)
    x = self.flatten(x)
    latent_code = self.fc(x)
    x = self.unfc(latent_code).view(-1, 16, 64, 3)
    reconstruction = self.decoder(x)

    return reconstruction, latent_code

```

```

criterion = F.mse_loss

autoencoder = Autoencoder().to(device)

optimizer = torch.optim.Adam(autoencoder.parameters(), lr=0.001)

```

▼ 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 = 60
train_losses = []
val_losses = []
latents = torch.Tensor()
truth = torch.Tensor()
pred = torch.Tensor()

for epoch in tqdm(range(n_epochs)):
    autoencoder.train()

```

```
train_losses_per_epoch = []
for i, X_batch in enumerate(data_tr):
    optimizer.zero_grad()
    reconstructed, latent_image = autoencoder(X_batch.to(device))
    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))

autoencoder.eval()
val_losses_per_epoch = []
with torch.no_grad():
    for X_batch in data_val:
        reconstructed, latent_image = autoencoder(X_batch.to(device))
        loss = criterion(reconstructed, X_batch.to(device))
        val_losses_per_epoch.append(loss.item())
        if epoch == n_epochs - 1:
            truth = torch.cat((truth, X_batch), 0)
            latents = torch.cat((latents, latent_image.to('cpu')), 0)
            pred = torch.cat((pred, reconstructed.to('cpu')), 0)

val_losses.append(np.mean(val_losses_per_epoch))

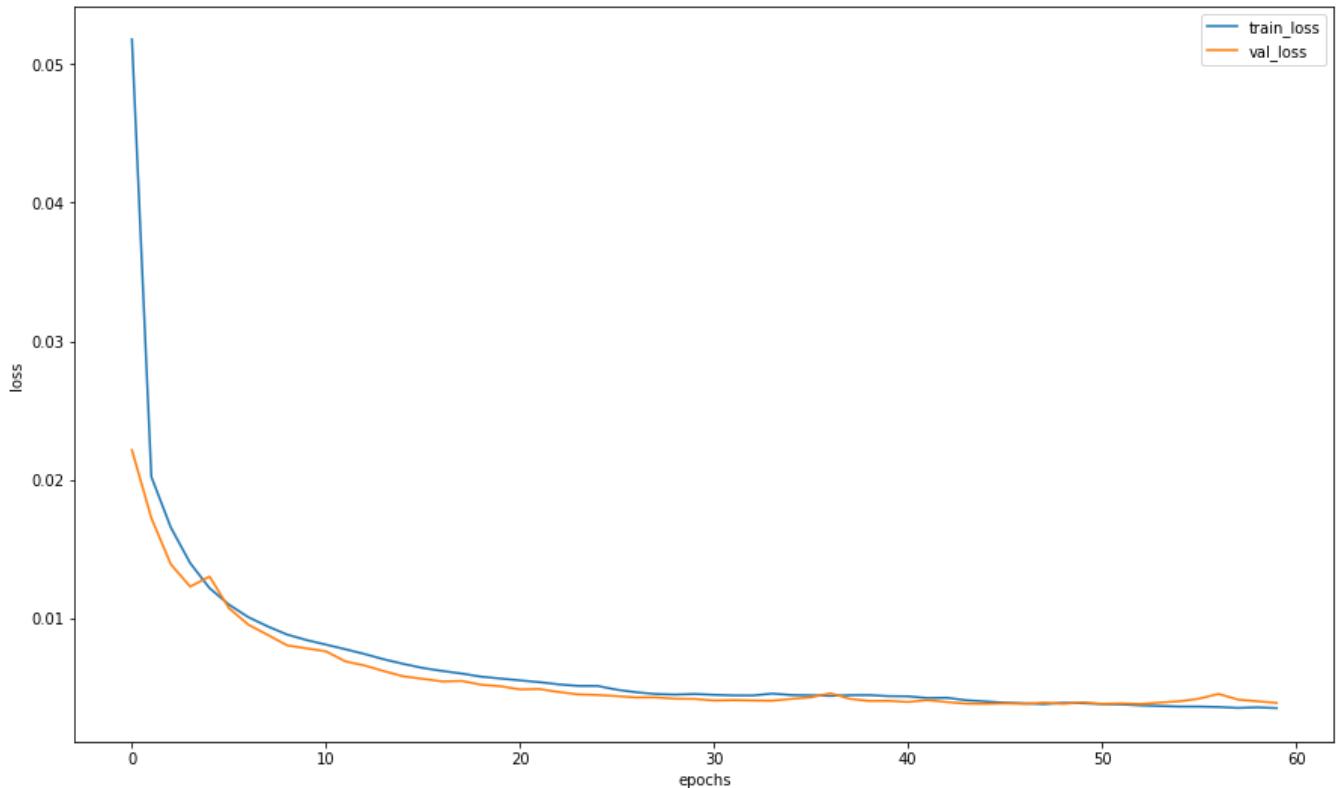
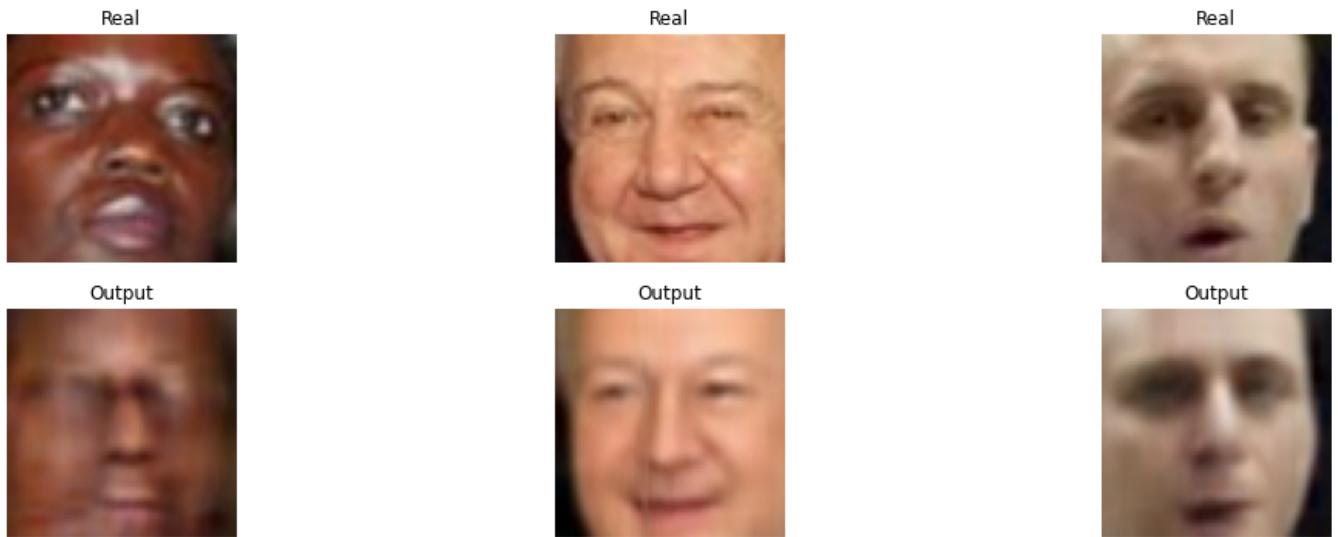
# Visualize tools
plt.figure(figsize=(18, 6))
clear_output(wait=True)
for k in range(3):
    plt.subplot(2, 3, k+1)
    plt.imshow(X_batch[k].cpu())
    plt.title('Real')
    plt.axis('off')

    plt.subplot(2, 3, k+4)
    plt.imshow(reconstructed[k].cpu())
    plt.title('Output')
    plt.axis('off')
plt.suptitle('%d / %d - loss: %f' % (epoch+1, n_epochs, loss.item()))
plt.show()

plt.figure(figsize=(15, 9))
plt.plot(train_losses, label="train_loss")
plt.plot(val_losses, label="val_loss")
plt.legend(loc='best')
plt.xlabel("epochs")
plt.ylabel("loss")
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for float32) for float32
Clipping input data to the valid range for imshow with RGB data ([0..1] for float32)

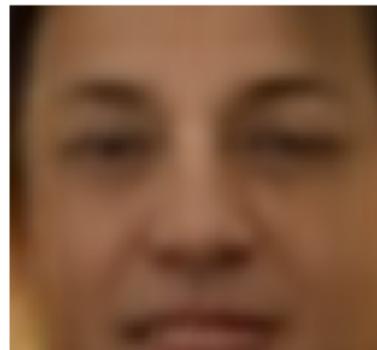
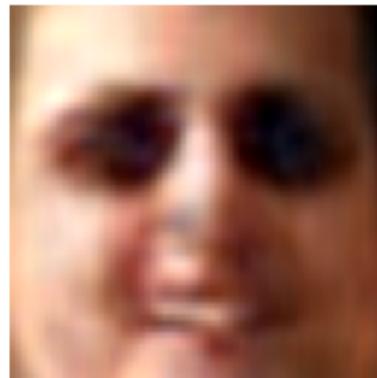
60 / 60 - loss: 0.004089



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

```
plt.figure(figsize=(8, 20))
for i, (gt, res) in enumerate(zip(truth[:5], pred[:5])):
    plt.subplot(5, 2, 2*i+1)
    plt.axis('off')
    plt.imshow(gt)
    plt.subplot(5, 2, 2*i+2)
    plt.axis('off')
    plt.imshow(res)
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for float32) for float32
Clipping input data to the valid range for imshow with RGB data ([0..1] for float32) for float32
Clipping input data to the valid range for imshow with RGB data ([0..1] for float32) for float32
Clipping input data to the valid range for imshow with RGB data ([0..1] for float32) for float32



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 фотографий. Так что в таком случае придется рандом немножко подогнать.

```
mu = torch.mean(latents, (0, 1)).item()
sigma = torch.std(latents, (0, 1)).item()
```

```
[REDACTED]  
[REDACTED]  
z = np.array([sigma * np.random.normal(0, 1, dim_code) + mu for i in range(25)])  
[REDACTED]  
[REDACTED]  
x = autoencoder.unfc(torch.FloatTensor(z).to(device)).view(-1, 16, 64, 3)  
output = autoencoder.decoder(x)  
  
# сгенерируем 25 рандомных векторов размера latent_space  
# z = np.array([np.random.normal(0, 1, 16) for i in range(25)])  
# output = autoencoder.unfc(latent_code).view(-1, dim_code, 64, 3)  
# output = autoencoder.decoder(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(x[i].cpu().detach().numpy())  
  
    plt.subplot(2, 6, i + 7)  
    plt.axis("off")  
    plt.imshow(output[i].cpu().detach().numpy())  
plt.show();
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for float type)
Clipping input data to the valid range for imshow with RGB data ([0..1] for float type)
Clipping input data to the valid range for imshow with RGB data ([0..1] for float type)
Clipping input data to the valid range for imshow with RGB data ([0..1] for float type)
Clipping input data to the valid range for imshow with RGB data ([0..1] for float type)
Clipping input data to the valid range for imshow with RGB data ([0..1] for float type)
Clipping input data to the valid range for imshow with RGB data ([0..1] for float type)
Clipping input data to the valid range for imshow with RGB data ([0..1] for float type)
Clipping input data to the valid range for imshow with RGB data ([0..1] for float type)
Clipping input data to the valid range for imshow with RGB data ([0..1] for float type)



- ▼ 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!

```
def to_latent_mean(x):
    x = autoencoder.encoder(x)
    x = autoencoder.flatten(x)
    x = autoencoder.fc(x).to('cpu')
```

```
z = torch.mean(x)
return x, z

def change(latent_target_mean, latent_orig_mean, latent_orig):
    vector = latent_target_mean - latent_orig_mean
    changed_latent_orig = latent_orig + vector
    x = autoencoder.unfc(changed_latent_orig.to(device)).view(-1, 16, 64, 3)
    changed_image = autoencoder.decoder(x)
    return changed_image

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

        plt.subplot(2, 6, i + 7)
        plt.axis("off")
        plt.imshow(y[i].cpu().detach().numpy())
    plt.show();

smile_idx = attrs.sort_values(['Smiling']).tail(15).index.tolist()
sad_idx = attrs.sort_values(['Smiling']).head(15).index.tolist()
smile_data = torch.FloatTensor(data[smile_idx]).to(device)
sad_data = torch.FloatTensor(data[sad_idx]).to(device)

latent_smile, latent_smile_mean = to_latent_mean(smile_data)
latent_sad, latent_sad_mean = to_latent_mean(sad_data)

changed_data = change(latent_smile_mean, latent_sad_mean, latent_sad)

show_images(sad_data, changed_data)
```



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`:

```
attrs.columns
```

```
Index(['Male', 'Asian', 'White', 'Black', 'Baby', 'Child', 'Youth',
       'Middle Aged', 'Senior', 'Black Hair', 'Blond Hair', 'Brown Hair',
       'Bald', 'No Eyewear', 'Eyeglasses', 'Sunglasses', 'Mustache', 'Smiling',
       'Frowning', 'Chubby', 'Blurry', 'Harsh Lighting', 'Flash',
       'Soft Lighting', 'Outdoor', 'Curly Hair', 'Wavy Hair', 'Straight Hair',
       'Receding Hairline', 'Bangs', 'Sideburns', 'Fully Visible Forehead',
       'Partially Visible Forehead', 'Obstructed Forehead', 'Bushy Eyebrows',
       'Arched Eyebrows', 'Narrow Eyes', 'Eyes Open', 'Big Nose',
       'Pointy Nose', 'Big Lips', 'Mouth Closed', 'Mouth Slightly Open',
       'Mouth Wide Open', 'Teeth Not Visible', 'No Beard', 'Goatee',
       'Round Jaw', 'Double Chin', 'Wearing Hat', 'Oval Face', 'Square Face',
       'Round Face', 'Color Photo', 'Posed Photo', 'Attractive Man',
       'Attractive Woman', 'Indian', 'Gray Hair', 'Bags Under Eyes',
       'Heavy Makeup', 'Rosy Cheeks', 'Shiny Skin', 'Pale Skin',
       '5 o\' Clock Shadow', 'Strong Nose-Mouth Lines', 'Wearing Lipstick',
       'Flushed Face', 'High Cheekbones', 'Brown Eyes', 'Wearing Earrings',
       'Wearing Necktie', 'Wearing Necklace'],
      dtype='object')
```

```
attrib = 'Child'
```

```
targ_idx = attrs.sort_values([attrib]).tail(15).index.tolist()
orig_idx = attrs.sort_values([attrib]).head(15).index.tolist()
targ_data = torch.FloatTensor(data[targ_idx]).to(device)
orig_data = torch.FloatTensor(data[orig_idx]).to(device)
```

```
latent_targ, latent_targ_mean = to_latent_mean(targ_data)
latent_orig, latent_orig_mean = to_latent_mean(orig_data)
```

```
changed_data = change(latent_targ_mean, latent_orig_mean, latent_orig)
```

```
show_images(orig_data, changed_data)
```



▼ Часть 2: Variational Autoencoder (10 баллов)

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

```
batch_size = 32
features = 8
# MNIST Dataset
train_dataset = datasets.MNIST(root='./mnist_data/', train=True, transform=transforms
test_dataset = datasets.MNIST(root='./mnist_data/', train=False, transform=transforms

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

```
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
9913344/? [00:00<00:00, 24588296.22it/s]
Extracting ./mnist_data/MNIST/raw/train-images-idx3-ubyte.gz to ./mnist_data/MNIST
```

▼ 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!

```
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./mnist
flat_size = features * 28 * 28
Extracting ./mnist_data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./mnist_data/MNIST

class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()

        self.flatten = nn.Flatten()

    #encoder
    self.encoder = nn.Sequential(
        nn.Conv2d(1, 64, kernel_size=5, stride=1, padding=2),
        nn.BatchNorm2d(64),
        nn.ReLU(),
        nn.Conv2d(64, features, kernel_size=5, stride=1, padding=2),
        nn.BatchNorm2d(features),
        nn.ReLU(),
        nn.Flatten(),
        nn.Linear(flat_size, 512),
        nn.ReLU(),
        nn.Linear(512, features * 2)
    )

    # decoder
    self.decoder = nn.Sequential(
        nn.Linear(features, 512),
        nn.ReLU(),
        nn.Linear(512, flat_size),
        nn.ReLU(),
        nn.Unflatten(1, (features, 28, 28)),
        nn.ConvTranspose2d(features, 64, kernel_size=5, stride=1, padding=2),
        nn.BatchNorm2d(64),
        nn.ReLU(),
        nn.ConvTranspose2d(64, 1, kernel_size=5, stride=1, padding=2),
    )

    def encode(self, xx):
        x = self.encoder(xx).view(xx.shape[0], 2, -1)
```

```

mu = x[:, 0, :]
logsigma = x[:, 1, :]

return mu, logsigma

def gaussian_sampler(self, mu, logsigma):
    if self.training:
        std = torch.exp(0.5 * logsigma) # 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
    else:
        # на инференсе возвращаем не случайный вектор из нормального распределени
        # на инференсе выход автоэнкодера должен быть детерминирован.
        return mu

def decode(self, z):
    x = self.decoder(z)
    reconstruction = torch.sigmoid(x)

    return reconstruction

def forward(self, x):
    mu, logsigma = self.encode(x)
    z = self.gaussian_sampler(mu, logsigma)
    x = self.decoder(z)
    reconstruction = torch.sigmoid(x)
    return reconstruction, mu, logsigma

```

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 разн
    """

```

```

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 обычн
    """
    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)

```

И обучим модель:

```

criterion = loss_vae

autoencoder = VAE().to(device)

optimizer = torch.optim.Adam(autoencoder.parameters())

n_epochs = 50
train_losses = []
test_losses = []
mus = torch.Tensor()
labels = torch.Tensor()

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

    train_losses.append(np.mean(train_losses_per_epoch))

    autoencoder.eval()
    test_losses_per_epoch = []
    with torch.no_grad():
        for batch, label in test_loader:
            reconstruction, mu, logsigma = autoencoder(batch.to(device))
            loss = criterion(batch.to(device).float(), mu, logsigma, reconstruction)
            test_losses_per_epoch.append(loss.item())
            if epoch == n_epochs - 1:
                mus = torch.cat((mus, mu.to('cpu')), 0)

```

```

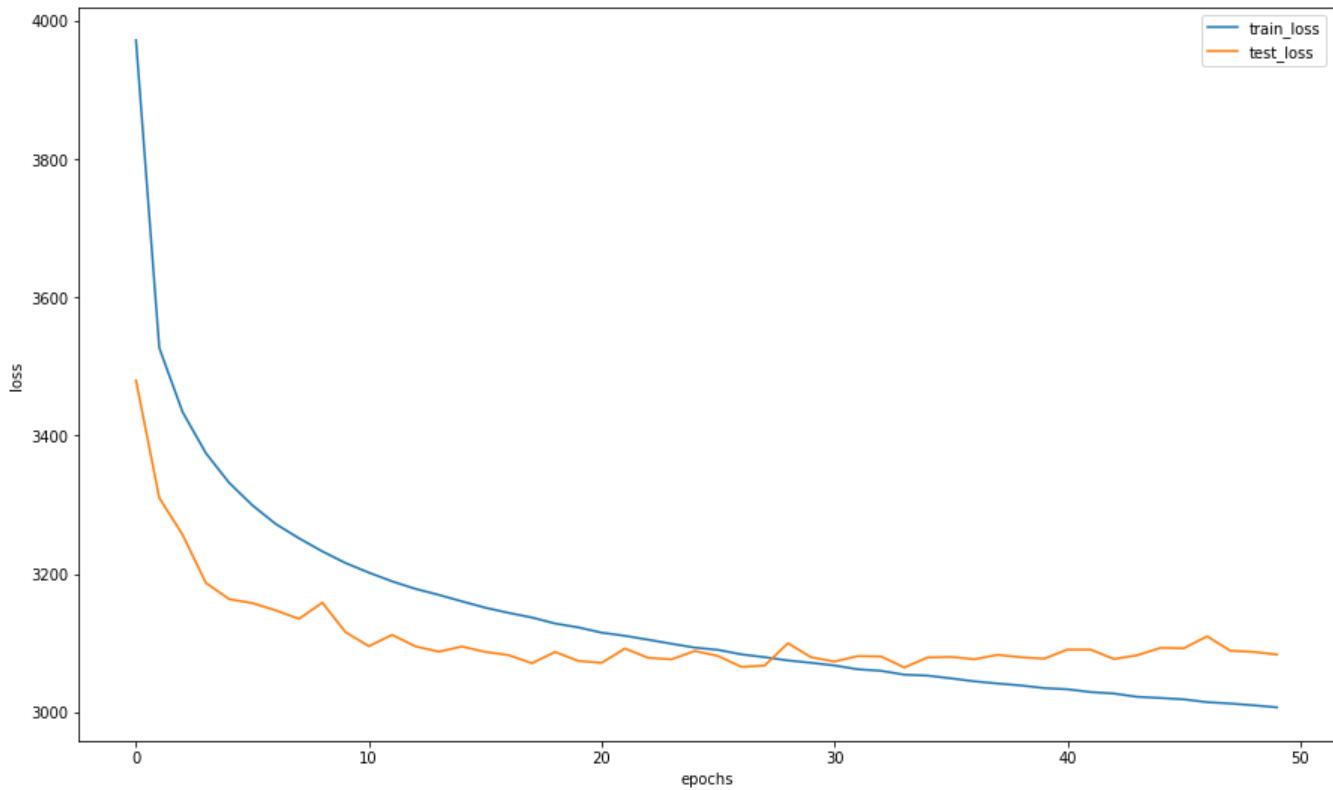
labels = torch.cat((labels, label), dim=0)

test_losses.append(np.mean(test_losses_per_epoch))

# Visualize tools
clear_output(wait=True)

plt.figure(figsize=(15, 9))
plt.plot(train_losses, label="train_loss")
plt.plot(test_losses, label="test_loss")
plt.legend(loc='best')
plt.xlabel("epochs")
plt.ylabel("loss")
plt.show()

```



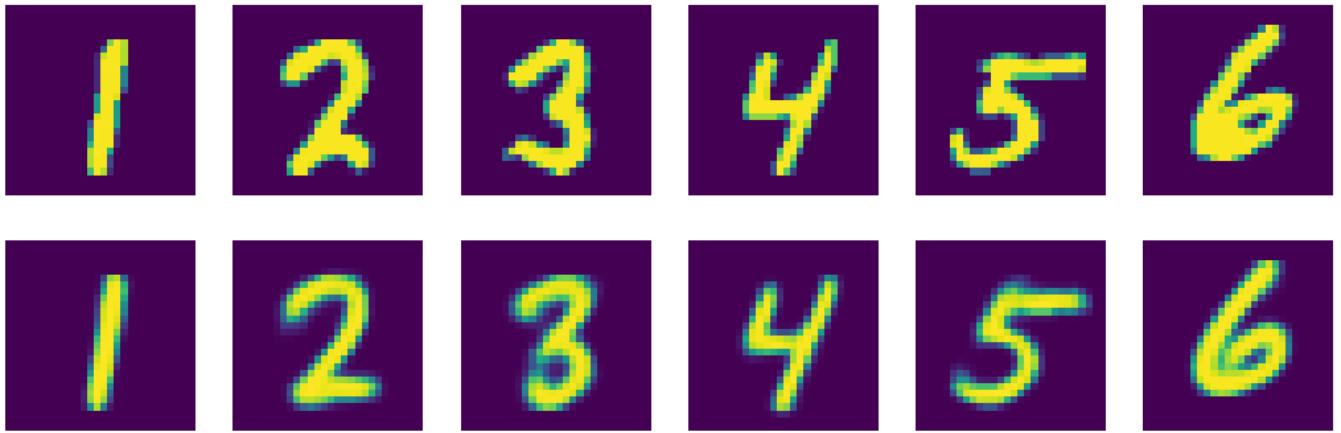
Давайте посмотрим, как наш тренированный VAE кодирует и восстанавливает картинки:

```

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

```

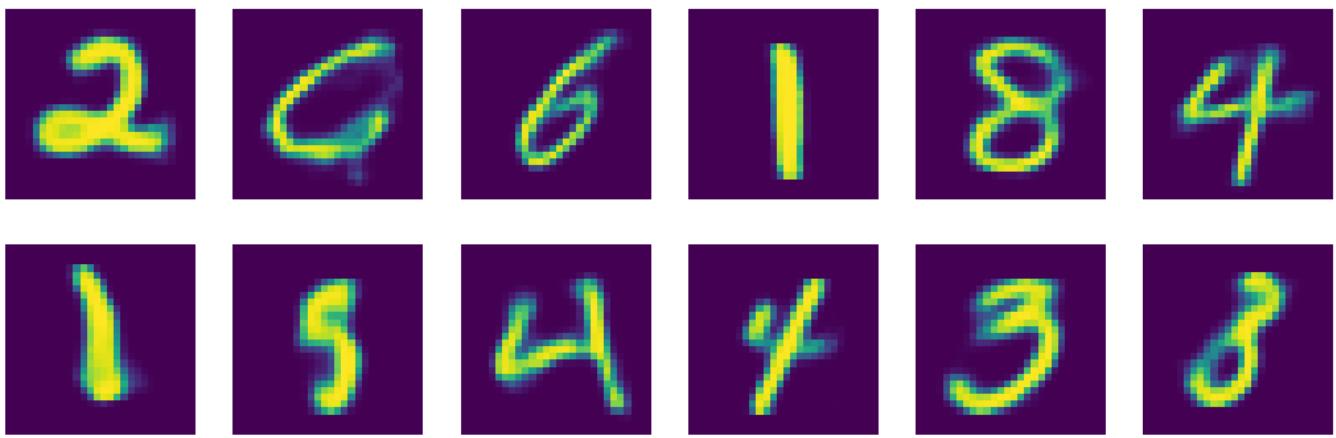
```
plt.imshow(batch[i][0].cpu().detach().numpy())
plt.subplot(2, 6, i + 7)
plt.axis("off")
plt.imshow(reconstruction[i][0].cpu().detach().numpy())
plt.show();
```



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
z = np.array([np.random.normal(0, 1, features) for i in range(12)])
output = autoencoder.decode(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][0].cpu().detach().numpy())
    plt.subplot(2, 6, i + 7)
    plt.axis("off")
    plt.imshow(output[i + 6][0].cpu().detach().numpy())
plt.show();
```



▼ 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 с (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.

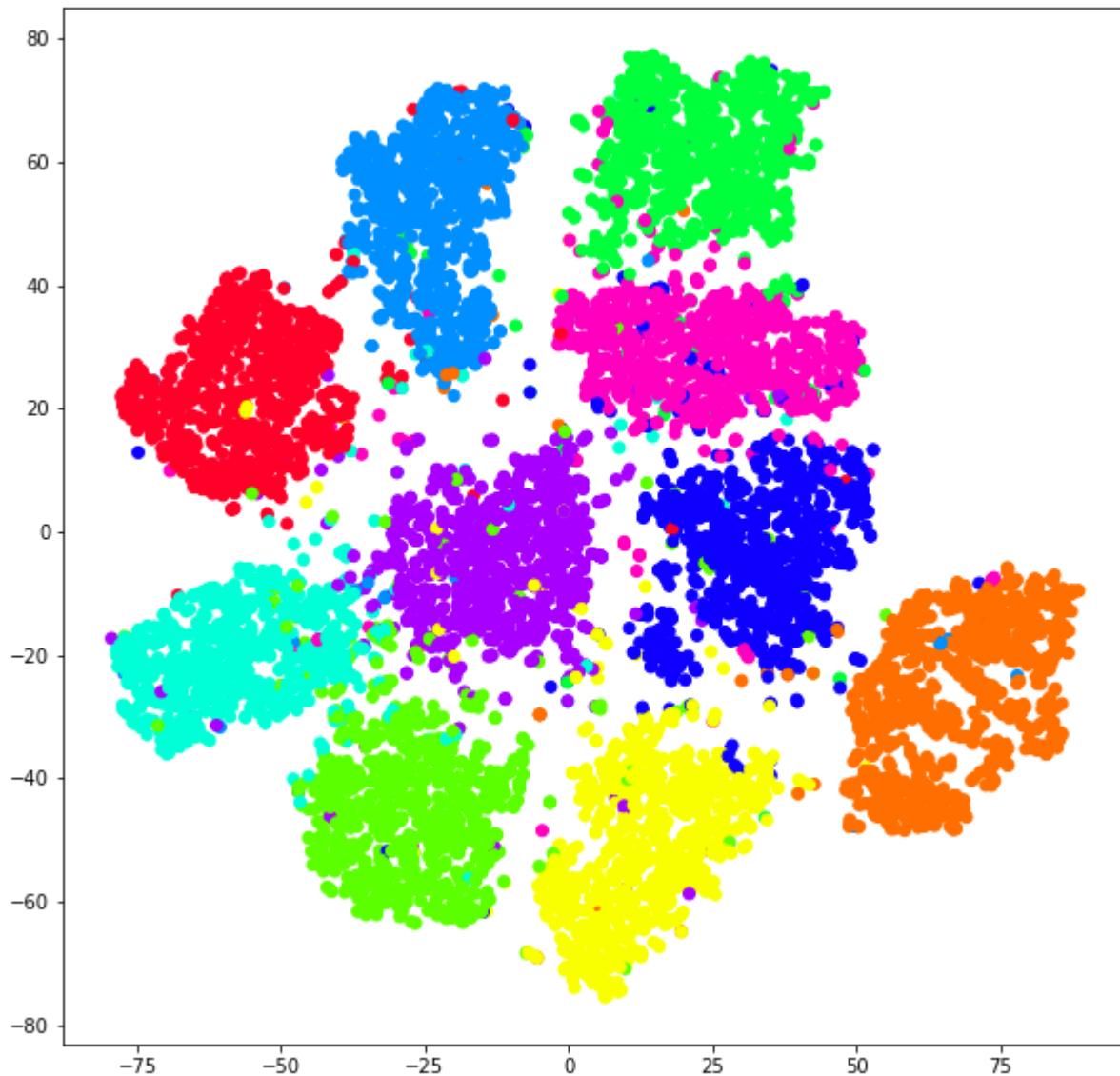
```
from sklearn.manifold import TSNE
```

```
mus_embedded = torch.from_numpy(TSNE(n_components=2).fit_transform(mus))
labels = labels.view(-1, 1)
mus_emb = torch.cat((mus_embedded, labels), dim = 1)

/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning,
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning,
  FutureWarning,
```

```
plt.figure(figsize=(10, 10))
plt.scatter(mus_emb[:, 0], mus_emb[:, 1], c=mus_emb[:, 2], cmap='gist_rainbow')

<matplotlib.collections.PathCollection at 0x7faff0a670d0>
```



Что вы думаете о виде латентного представления?

Congrats v2.0!

► 2.3. Conditional VAE (6 баллов)

▶ Скрыто 16 ячеек.

► BONUS 1: Denoising

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

[] ↴ Скрыто 7 ячеек.

‣ BONUS 2: Image Retrieval

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

[] ↴ Скрыто 8 ячеек.

❗ 0 сек. выполнено в 13:17

● ✕