Autoencoders

В этом ноутбуке мы будем тренировать автоэнкодеры кодировать лица людей. Для этого возьмем следующий датасет: "Labeled Faces in the Wild" (LFW) (http://vis-www.cs.umass.edu/lfw/). Код для скачивания и загрузки датасета написан за вас в файле get_dataset.py

Vanilla Autoencoder (2 балла)

Prepare the data

```
import numpy as np
from torch.autograd import Variable
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

import torchvision.transforms as transforms
import torchvision.models as models
from PIL import Image

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


# The following line fetches you two datasets: images, usable for autoencoder training and
# Those attributes will be required for the final part of the assignment (applying smiles)
from get_dataset import fetch_dataset
data, attrs = fetch_dataset()
```

```
images not found, donwloading...
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:

```
attrs.head()
```

	Male	Asian	White	Black	Baby	Child	Youth	Middle Aged
0	1.56835	-1.88904	1.7372	-0.929729	-1.4718	-0.19558	-0.835609	-0.351468
1	0.169851	-0.982408	0.422709	-1.28218	-1.36006	-0.867002	-0.452293	-0.197521
2	0.997749	-1.36419	-0.157377	-0.756447	-1.89183	-0.871526	-0.862893	0.0314447
3	1.12272	-1.9978	1.91614	-2.51421	-2.58007	-1.40424	0.0575511	0.000195882
4	1.07821	-2.0081	1.67621	-2.27806	-2.65185	-1.34841	0.649089	0.0176564

```
def isNaN(num): return num != num

smile = []
smile_table = attrs['Smiling']
smile = smile_table.where(smile_table > 0.5)
without_smile = smile_table.where(smile_table < 0)


from sklearn.model_selection import train_test_split
X_train, X_test = train_test_split(data, random_state=42)


def plt_imshow(inp, title=None, plt_ax=plt, default=False):
    """Imshow для тензоров"""
    plt_ax.imshow(inp)
    if title is not None:
        plt_ax.set_title(title)
    plt_ax.grid(False)

fig, ax = plt.subplots(nrows=3, ncols=3,figsize=(8, 8), \
                       sharey=True, sharex=True)
for fig_x in ax.flatten():
    random_characters = int(np.random.uniform(0,1000))
    im_val= torch.from_numpy(X_train[random_characters])
    img_label = " "
    plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)
```

```
batch_size = 128
X_train = torch.from_numpy(X_train)
X_test = torch.from_numpy(X_test)
train_dataloader = torch.utils.data.DataLoader(X_train, batch_size=batch_size, drop_last=T
test_dataloader = torch.utils.data.DataLoader(X_test, batch_size=1, shuffle=True)
```

```
from torchvision import datasets, transforms
batch_size = 128
transform = transforms.Compose([
    transforms.Resize((28,28)),
    transforms.ToTensor(),
    transforms.Normalize((0.5), (0.5))
])

train_dataset_MNIST = datasets.MNIST('./data', transform=transform, download=True)
train_dataloader_MNIST = torch.utils.data.DataLoader(train_dataset_MNIST, batch_size=batch

test_dataset_MNIST = datasets.MNIST('./data', transform=transform, download=True, train=Fa
test_dataloader_MNIST = torch.utils.data.DataLoader(test_dataset_MNIST, batch_size=1, shuf
```

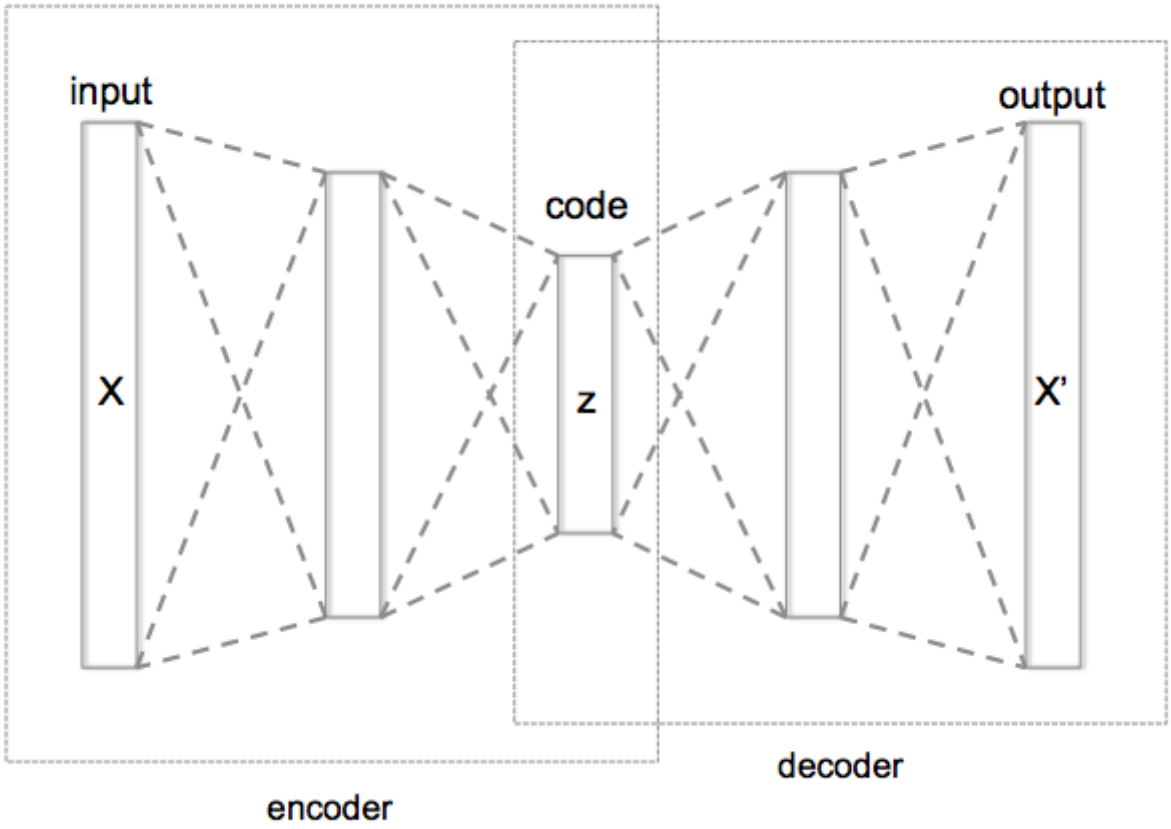
```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MN
                                                                9920512/? [00:20<00:00, 2791071.69it/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw
```

Autoencoder

В этом разделе мы напишем и обучем обычный автоэнкодер.

Надеюсь, что к этому моменту вы уже почитали про автоэнкодеры и знаете, зачем они нужны и какова их архитектура. Если нет, то начните с этих ссылок:

https://habr.com/ru/post/331382/
https://towardsdatascience.com\intuitively-understanding-variational-autoencoders-1bfe67eb5daf



^ вот так выглядит автоэнкодер

```
dim_code = 100 # выберите размер латентного вектора, т.е. code, самой "узкой" части автоэн
```

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

```
from copy import deepcopy
```

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

            self.encoder = nn.Sequential(
                nn.Linear(45*45*3,1500),
                nn.BatchNorm1d(1500),
                nn.ReLU(),
                nn.Linear(1500,1000),
                nn.BatchNorm1d(1000),
                nn.ReLU(),
                nn.Linear(1000, dim_code),
                nn.BatchNorm1d(dim_code),
                nn.ReLU()
                # nn.Linear(28*28, 512),
                # nn.ReLU(),
                # nn.Linear(512, 256),
                # nn.ReLU(),
                # nn.Linear(256, dim_code),
                # #nn.ReLU()
                # nn.Sigmoid()
                # nn.Conv2d(3, 16, 3, padding=1),
                # nn.ReLU(),
                # nn.Conv2d(16, 32, 3, padding=1),
                # nn.ReLU(),
                # nn.Conv2d(32, 64, 7)
            )

            self.decoder = nn.Sequential(
                nn.Linear(dim_code,1000),
                nn.BatchNorm1d(1000),
                nn.ReLU(),
                #nn.Linear(500,1000),
                #nn.ReLU(),
                nn.Linear(1000,1500),
                nn.BatchNorm1d(1500),
                nn.ReLU(),
                nn.Linear(1500,45*45*3),
                # nn.Linear(dim_code, 256),
                # nn.ReLU(),
                # nn.Linear(256, 512),
                # nn.ReLU(),
                # nn.Linear(512, 28*28),
                # #nn.Sigmoid()
                # nn.ReLU()
                # nn.ConvTranspose2d(64, 32, 7),
                # nn.ReLU(),
                # nn.ConvTranspose2d(32, 16, 3, padding=1),
                # nn.ReLU(),
                # nn.ConvTranspose2d(16, 3, 3, padding=1),
                nn.Sigmoid()
            )
            #<определите архитектуры encoder и decoder>

        def forward(self, x):
```

```
        latent_code = self.encoder(x)
        reconstruction = self.decoder(latent_code)
        # <реализуйте forward проход автоэнкодера
        # в качестве ваозвращаемых переменных -- латентное представление картинки (latent_
        # и полученная реконструкция изображения (reconstruction)>
        return reconstruction, latent_code

    def reconst(self, latent):
        return self.decoder(latent)

    def latent(self, x):
        return self.encoder(x)


criterion = torch.nn.CrossEntropyLoss()

autoencoder = Autoencoder()

optimizer = 'adam'
```

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 =)

```
def train(net, train_dataloader, epochs=5, flatten=False, loss_fn=nn.BCELoss(), title=None
    optim = torch.optim.Adam(net.parameters(), lr=learning_rate, weight_decay=1e-4)

    train_losses = 0
    epoch = 1
    history = []

    for i in range(epochs):
        for y_true in train_dataloader:
            y_true = y_true.to(device)
            y_true = y_true.view(y_true.size(0), 45*45*3)
            # ==================forward====================
            y_pred, y_latent = net(y_true.float())
            loss = loss_fn(y_pred.float(), y_true.float())#.detach())
            # ==================backward===================
            optim.zero_grad()
            loss.backward()
            optim.step()
            train_losses = loss.item()
            history.append(loss.item())
        print('Epoch: {} \tTraining Loss: {:.6f}'.format(epoch, train_losses))
        epoch += 1

    return history
# <тут Ваш код тренировки автоэнкодера>
```

```
autoencoder = Autoencoder().to(device)
history = train(autoencoder, train_dataloader, epochs=15, flatten=True, title='Autoencoder
```

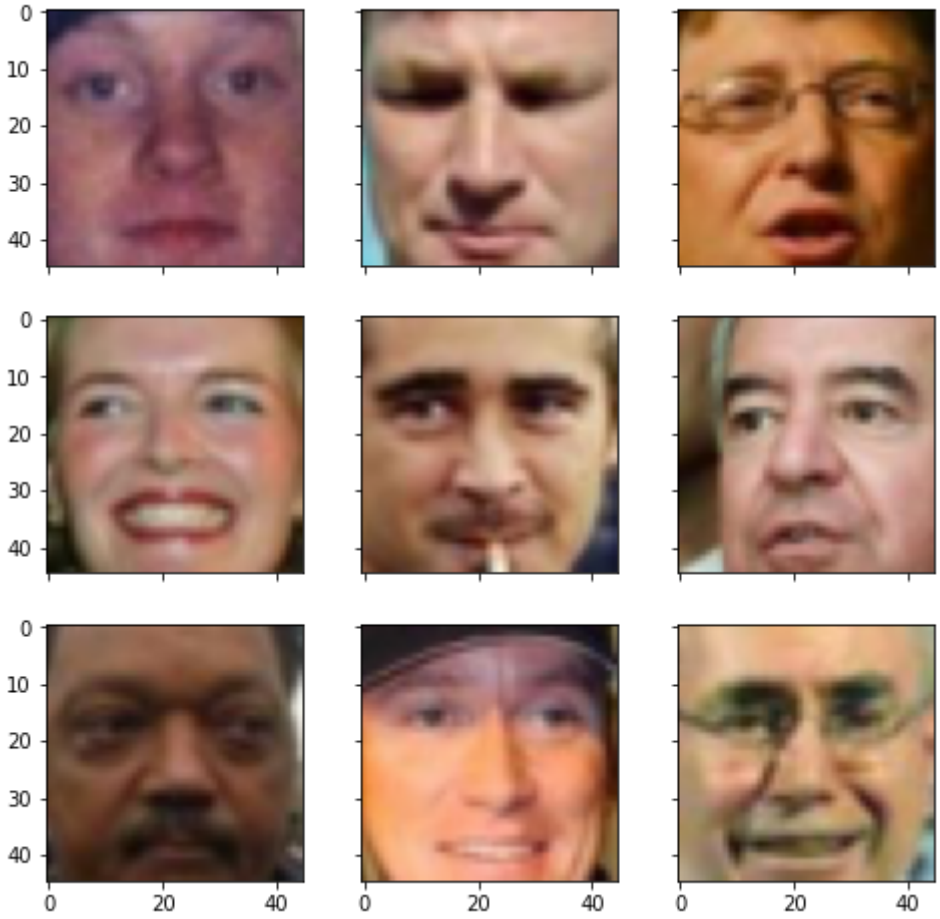
```
Epoch: 1        Training Loss: 0.622827
Epoch: 2        Training Loss: 0.616511
Epoch: 3        Training Loss: 0.612477
Epoch: 4        Training Loss: 0.611767
Epoch: 5        Training Loss: 0.614718
Epoch: 6        Training Loss: 0.606389
Epoch: 7        Training Loss: 0.608854
Epoch: 8        Training Loss: 0.614543
Epoch: 9        Training Loss: 0.612160
Epoch: 10       Training Loss: 0.608697
Epoch: 11       Training Loss: 0.609436
Epoch: 12       Training Loss: 0.607888
Epoch: 13       Training Loss: 0.607707
Epoch: 14       Training Loss: 0.609082
Epoch: 15       Training Loss: 0.616358
```

```
train_loss = history
plt.figure(figsize=(15,10))
plt.plot(train_loss, label='Train loss')
plt.legend(loc='best')
plt.xlabel("epochs")
plt.ylabel("loss")
plt.plot();
```

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

```
rec_imgs = []
latents = []
true_face = []
i = 0
x = 9
autoencoder.eval()
for y_true in test_dataloader:
    if i == x:
      break
    true_face.append(y_true)
    y_true = y_true.view(y_true.size(0), 45*45*3)
    y_true = y_true.to(device)
    #y_true = y_true.reshape(1, 3, 45, 45)
    y_pred, y_latent = autoencoder.forward(y_true.float())
    y_pred = torch.reshape(y_pred, (45, 45, 3))

  #print(y_pred.shape)

    rec_imgs.append(y_pred)
    latents.append(y_latent)
    i += 1

fig0, ax0 = plt.subplots(nrows=3, ncols=3,figsize=(8, 8), \
                         sharey=True, sharex=True)
i = 0
for fig_x in ax0.flatten():
    im_val= torch.reshape(true_face[i], (45, 45, 3))
    img_label = " "
    i += 1
    plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)

fig, ax = plt.subplots(nrows=3, ncols=3,figsize=(8, 8), \
                         sharey=True, sharex=True)
i = 0
for fig_x in ax.flatten():
    im_val= rec_imgs[i]
    img_label = " "
    i += 1
    plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)

fig1, ax1 = plt.subplots(nrows=3, ncols=3,figsize=(8, 8), \
                         sharey=True, sharex=True)
j = 0
for fig_x in ax1.flatten():
    im_val = torch.reshape(latents[j], (10, 10))
```

```
        img_label = " "
        j += 1
        plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)
```



Not bad, right?

Sampling

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

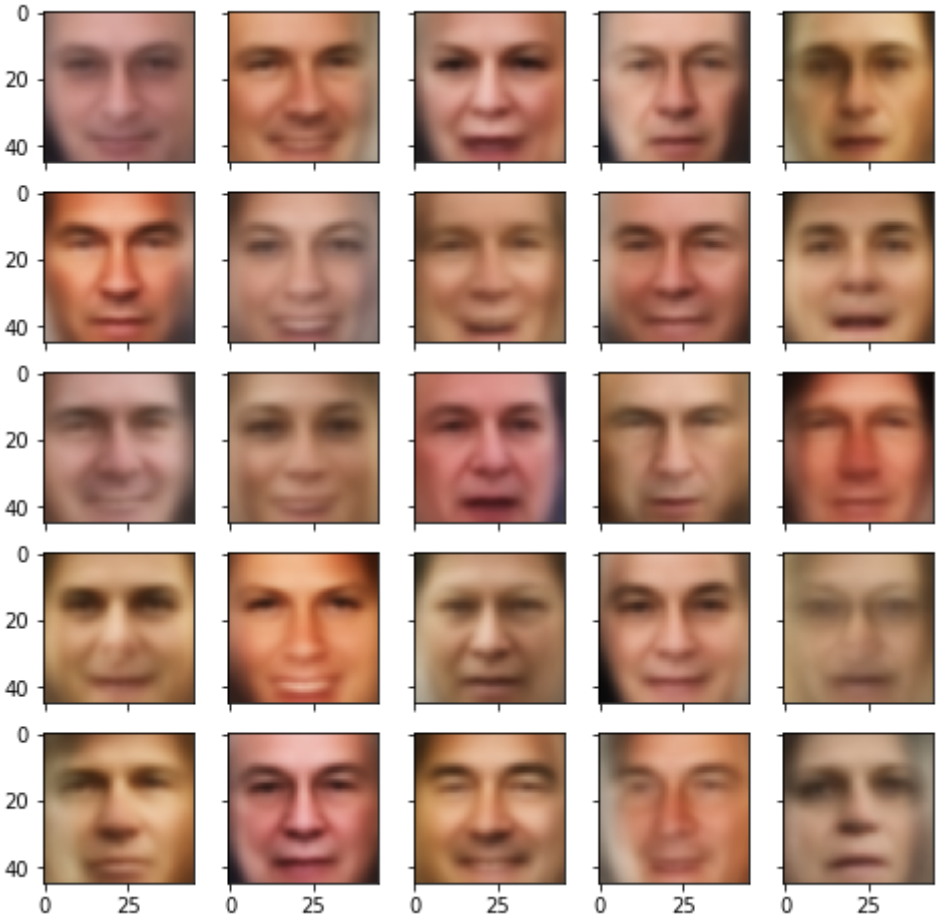
Давайте возьмем и подсунем декодеру какие-нибудь сгенерированные нами векторы (например, из нормального распределения) и посмотрим на результат реконструкции декодера:

If that doesn't work

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,). А чтобы у нас получались лица при запихивании вектора декодеру, вектор должен быть распределен так же, как лаьентные векторы реальных фоток. Так что ридется рандом подогнать.

```
# сгенерируем 25 рандомных векторов размера latent_space
z = np.random.randn(25, dim_code)
z = torch.from_numpy(z)
```

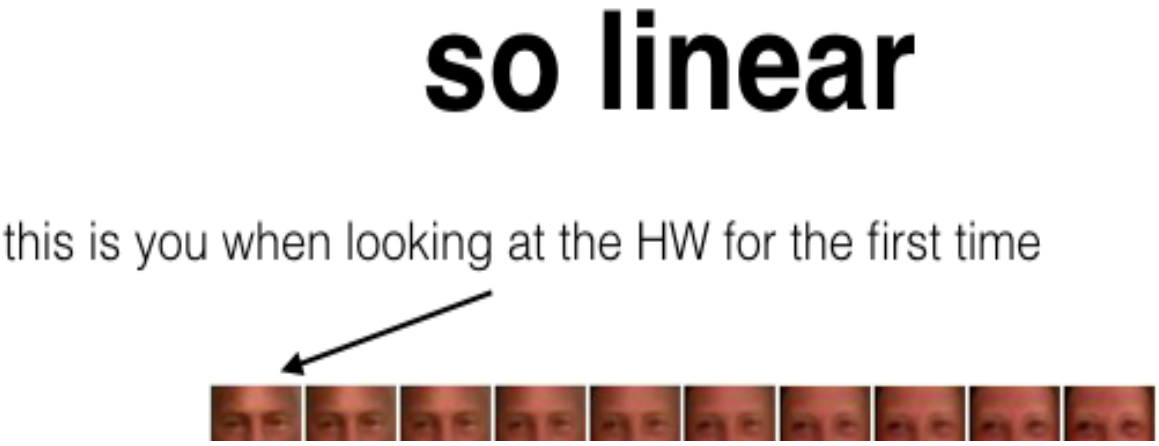
```
output = autoencoder.reconst(z.float())

fig2, ax2 = plt.subplots(nrows=5, ncols=5,figsize=(8, 8), \
                         sharey=True, sharex=True)
j = 0
for fig_x in ax2.flatten():
  im_val = torch.reshape(output[j], (45, 45, 3))
  img_label = " "
  j += 1
  plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)
# <выведите тут полученные картинки>
```



Congrats!

Time to make fun!

Давайте научимся пририсовывать людям улыбки =)



План такой:

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

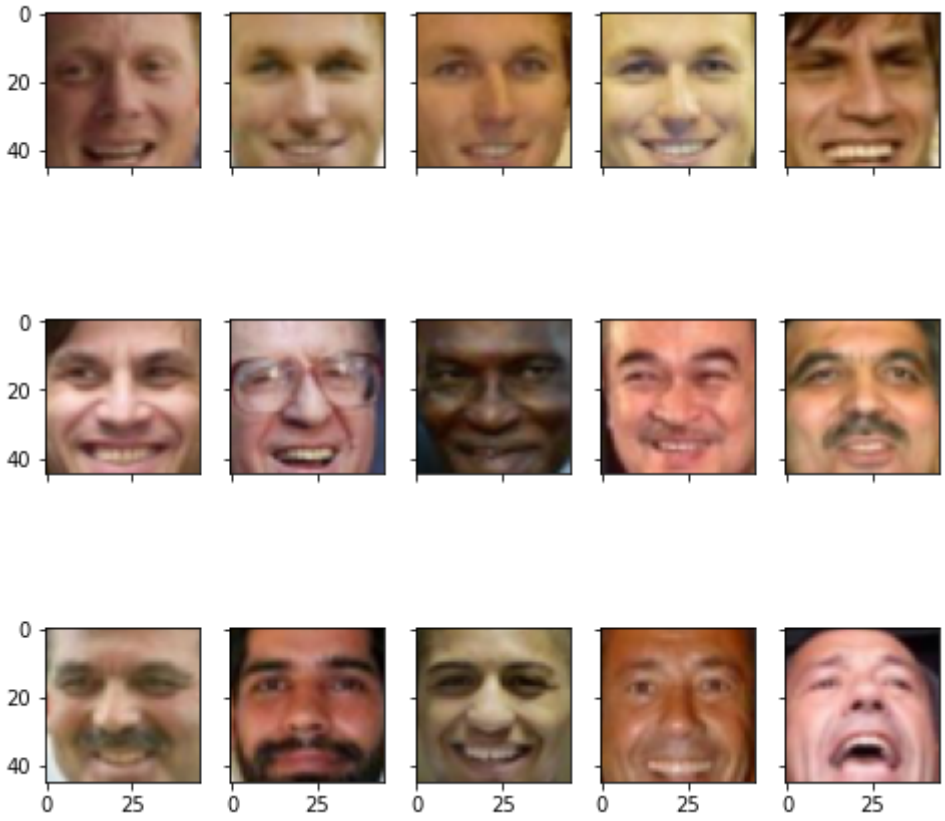
Найти людей с улыбками вам поможет файл с описанием датасета, скачанный вместе с датасетом. В нем указаны имена картинок и присутствубщие атрибуты (улыбки, очки...)

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

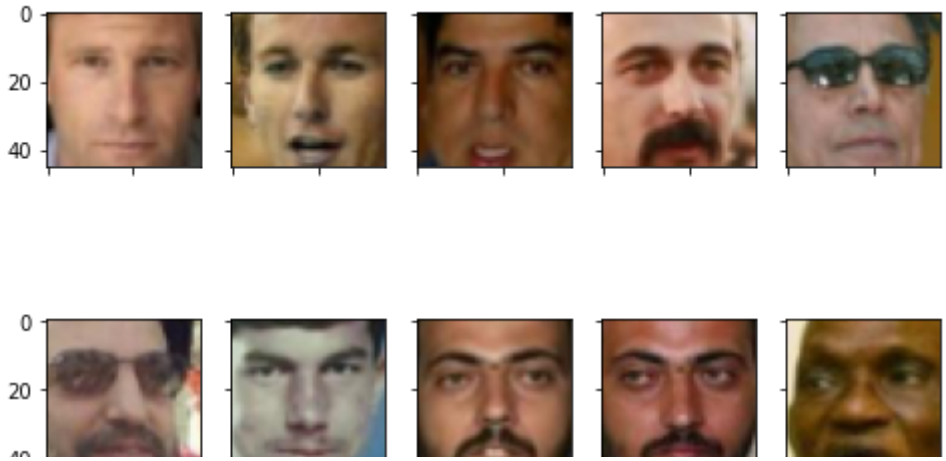
3) Вычислить, собственно, вектор улыбки -- посчитать разность между средним латентным вектором улыбающихся людей и средним латентным вектором грустных людей

3) А теперь приделаем улыбку грустному человеку: добавим полученный в пункте 3 вектор к латентному вектору грустного чувака и прогоним полученный вектор через decoder. Получим того же человека, но уже не грустненького!

```
smile_arr = []
w_smile_arr = []
count = 0
for i in range(len(smile)):
  if not isNaN(smile[i]):
    smile_arr.append(data[i])
    count += 1
  if count == 15:
    break
fig3, ax3 = plt.subplots(nrows=3, ncols=5,figsize=(8, 8), \
                         sharey=True, sharex=True)
j = 0
for fig_x in ax3.flatten():
  im_val = torch.from_numpy(smile_arr[j])
  img_label = " "
  j += 1
  plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)

# <а вот тут все это надо запрогать, да>
```




```
count = 0
for i in range(len(without_smile)):
  if not isNaN(without_smile[i]):
    w_smile_arr.append(data[i])
    count += 1
  if count == 15:
    break
w_smile_arr = w_smile_arr[0:15]
len(w_smile_arr)

fig4, ax4 = plt.subplots(nrows=3, ncols=5,figsize=(8, 8), \
                         sharey=True, sharex=True)

j = 0
for fig_x in ax4.flatten():
  im_val = torch.from_numpy(w_smile_arr[j])
  img_label = " "
  j += 1
  plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)
```




```
smile_arr = torch.from_numpy(np.array(smile_arr))
w_smile_arr = torch.from_numpy(np.array(w_smile_arr))
smile_arr = smile_arr.view(-1, 6075).to(device)# smile_arr.reshape(15, 3, 45, 45) #smile_a
w_smile_arr = w_smile_arr.view(-1, 6075).to(device)# w_smile_arr.reshape(15, 3, 45, 45) #w

latent_smile = autoencoder.latent(smile_arr.float())
latent_without_smile = autoencoder.latent(w_smile_arr.float())

latent_smile_mean = torch.mean(latent_smile)
latent_without_smile_mean = torch.mean(latent_without_smile)

latent_smile_vector = latent_smile_mean - latent_without_smile_mean

test_vec = latent_without_smile + latent_smile_vector

output = autoencoder.reconst(test_vec.float())

fig5, ax5 = plt.subplots(nrows=3, ncols=5,figsize=(8, 8), \
                         sharey=True, sharex=True)
j = 0
for fig_x in ax5.flatten():
  im_val = torch.reshape(output[j], (45, 45, 3))
  img_label = " "
  j += 1
  plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)
```

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 lwf_deepfinetuned.txt =)

Variational Autoencoder. (3 балла)

Представляю вам проапгрейдженную версию автоэнкодеров -- вариационные автоэнкодеры.

https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

```
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()
        self.fc1 = nn.Linear(28*28, 512)
        self.fc21 = nn.Linear(512, 100)
        self.fc22 = nn.Linear(512, 100)

        self.relu = nn.ReLU()

        self.fc3 = nn.Linear(100, 512)
        self.fc4 = nn.Linear(512, 28*28)
        # <определите архитектуры encoder и decoder
        # помните, у encoder должны быть два "хвоста",
        # т.е. encoder должен кодировать картинку в 2 переменные -- mu и logsigma>

    def encode(self, x):
        x = self.relu(self.fc1(x))
        mu = self.fc21(x)
        logsigma = self.fc22(x)
        # <реализуйте forward проход энкодера
        # в качестве ваозвращаемых переменных -- mu и logsigma>

        return mu, logsigma

    def gaussian_sampler(self, mu, logsigma):
```

```
        if self.training:
            std = torch.exp(0.5 *logsigma)
            eps = torch.randn_like(std)
            return eps.mul(std).add_(mu)
            # <засемплируйте латентный вектор из нормального распределения с параметрами m
        else:
            # на инференсе возвращаем не случайный вектор из нормального распределения, а
            # на инференсе выход автоэнкодера должен быть детерминирован.
            return mu

    def decode(self, z):
        z = self.relu(self.fc3(z))
        #torch.sigmoid(self.fc4(z))
        # <реализуйте forward проход декодера
        # в качестве ваозвращаемой переменной -- reconstruction>
        return torch.sigmoid(self.fc4(z))

    def forward(self, x):
        mu, logsigma = self.encode(x)
        z = self.gaussian_sampler(mu, logsigma)
        reconstruction = self.decode(z)
        # <используя encode и decode, реализуйте forward проход автоэнкодера
        # в качестве ваозвращаемых переменных -- mu, logsigma и reconstruction>
        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:

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

Формула для KL-дивергенции:

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

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

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

def log_likelihood(x, reconstruction):
    """
    часть функции потерь, которая отвечает за качество реконструкции (как mse в обычном au
    """
    loss = F.binary_cross_entropy(reconstruction, x, reduction='sum') #<binary_cross-entro
```

```
    loss = F.binary_cross_entropy(reconstruction, x, reduction='sum') #<binary cross entr
    return loss

def loss_vae(x, mu, logsigma, reconstruction):
    BCE = log_likelihood(x, reconstruction)
    KLD = KL_divergence(mu, logsigma)
    return (BCE + KLD)#<соедините тут две компоненты лосса. Mind the sign!>
```

И обучим модель:

```
criterion = loss_vae

autoencoder = Autoencoder()


from torchvision import datasets, transforms

transform = transforms.Compose([
    transforms.Resize((28,28)),
    transforms.ToTensor(),
    transforms.Normalize((0.5), (0.5))
])

train_dataset_MNIST = datasets.MNIST('./data', transform=transform, download=True)
train_dataloader_MNIST = torch.utils.data.DataLoader(train_dataset_MNIST, batch_size=batch

test_dataset_MNIST = datasets.MNIST('./data', transform=transform, download=True, train=Fa
test_dataloader_MNIST = torch.utils.data.DataLoader(test_dataset_MNIST, batch_size=1, shuf
#<обучите модель, как и autoencoder, но на датасете MNIST>
```

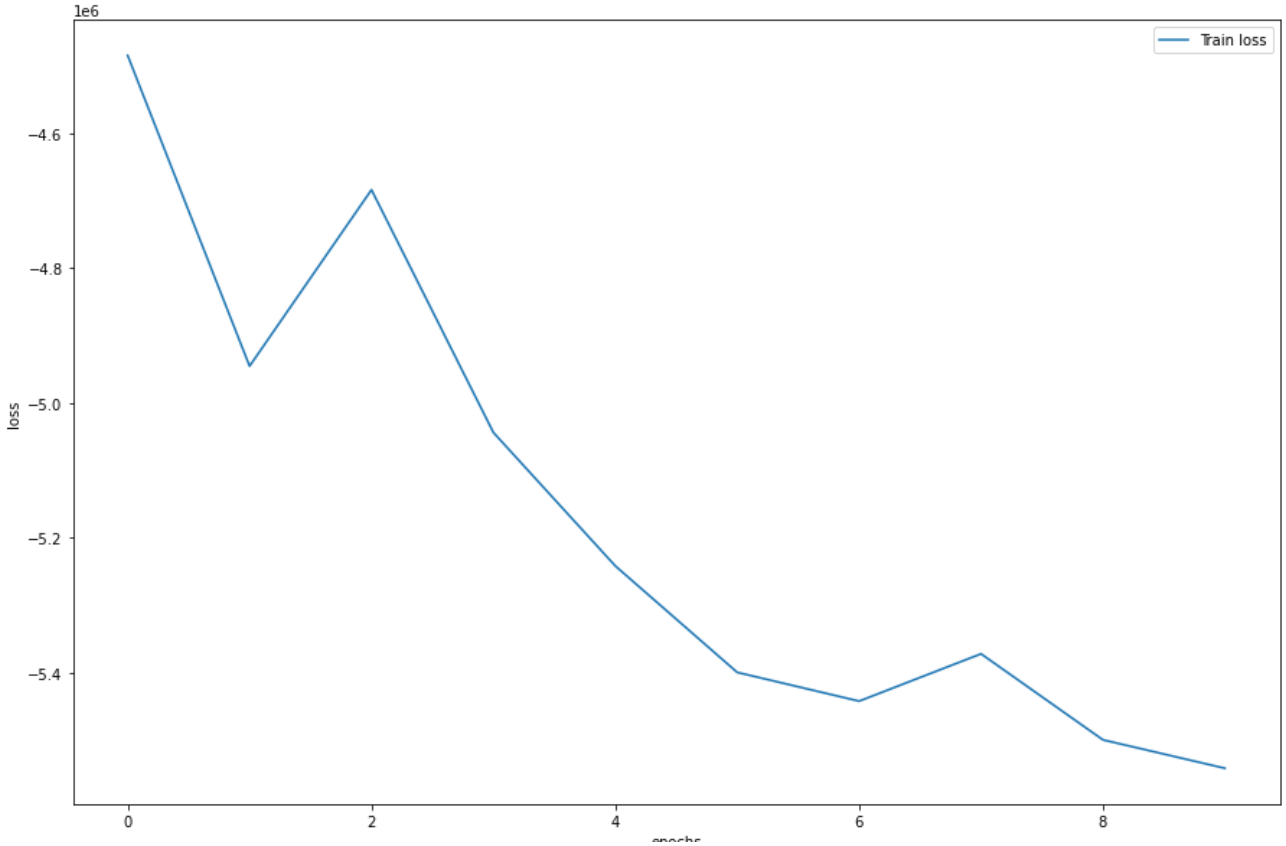
```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MN
89%                                 8863744/9912422 [00:01<00:00, 1463915.03it/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MN
0%                                  0/28881 [00:00<?, ?it/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNI
26%                                 434176/1648877 [00:00<00:04, 243520.18it/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNI
0%                                  0/4542 [00:00<?, ?it/s]
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
Processing...
/usr/local/lib/python3.6/dist-packages/torchvision/datasets/mnist.py:480: UserWarnin
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
Done!
```

```
def train_vae(net, train_dataloader, epochs=5, flatten=False, loss_fn=criterion, title=Nor
    optim = torch.optim.Adam(net.parameters())

    train_losses = 0
```

```
        epoch = 1
        history = []

        for i in range(epochs):
            for y_true, label in train_dataloader:
                y_true = y_true.to(device)
                y_true = y_true.view(y_true.size(0), 28*28)
                # ==================forward====================
                mu, logsigma, reconstruction = net(y_true.float())
                loss = loss_fn(y_true.float(), mu, logsigma, reconstruction)#.detach())
                # ==================backward===================
                optim.zero_grad()
                loss.backward()
                optim.step()

                train_losses = loss.item()
            print('Epoch: {} \tTraining Loss: {:.6f}'.format(epoch, train_losses))
            epoch += 1
            history.append(loss.item())

        return history
    # <тут Ваш код тренировки автоэнкодера>


    vae = VAE().to(device)
    vae_history = train_vae(vae, train_dataloader_MNIST, 10)
```

```
     Epoch: 1        Training Loss: -4485029.500000
     Epoch: 2        Training Loss: -4945295.000000
     Epoch: 3        Training Loss: -4684052.500000
     Epoch: 4        Training Loss: -5043616.000000
     Epoch: 5        Training Loss: -5241309.500000
     Epoch: 6        Training Loss: -5398972.500000
     Epoch: 7        Training Loss: -5441594.000000
     Epoch: 8        Training Loss: -5371517.500000
     Epoch: 9        Training Loss: -5498946.000000
     Epoch: 10       Training Loss: -5541026.500000
```

```
    train_loss = vae_history
    plt.figure(figsize=(15,10))
    plt.plot(train_loss, label='Train loss')
    plt.legend(loc='best')
    plt.xlabel("epochs")
    plt.ylabel("loss")
    plt.plot();
```



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

```
rec_imgs_MHIST = []
#latents_MHIST = []
true_face_MHIST = []
i = 0
x = 9
for y_true, _ in test_dataloader_MNIST:
  if i == x:
    break
  true_face_MHIST.append(y_true)
  y_true = y_true.view(y_true.size(0), 28*28).to(device)
  mu, logsigma, reconstruction = vae.forward(y_true.float())
  reconstruction = torch.reshape(reconstruction, (28, 28))

  #print(y_pred.shape)

  rec_imgs_MHIST.append(reconstruction)
  #latents.append(y_latent)
  i += 1

fig0, ax0 = plt.subplots(nrows=3, ncols=3,figsize=(8, 8), \
                         sharey=True, sharex=True)
i = 0
for fig_x in ax0.flatten():
  im_val= torch.reshape(true_face_MHIST[i], (28, 28))
  img_label = " "
  i += 1
  plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)

fig, ax = plt.subplots(nrows=3, ncols=3,figsize=(8, 8), \
```

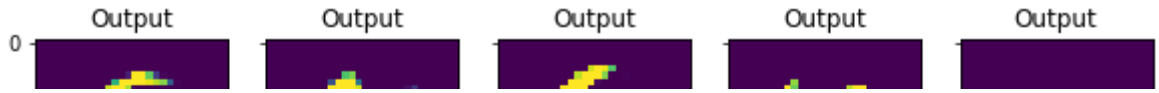
```
                          sharey=True, sharex=True)
    i = 0
    for fig_x in ax.flatten():
      im_val= rec_imgs_MHIST[i]
      img_label = " "
      i += 1
      plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)
    #< тут Ваш код: выведите первые X картинок и их реконструкций из val выборки на экран>
```

And finally sample from VAE.

Sampling

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 перем
z = np.array([np.random.normal(0, 1, 100) for i in range(10)])
z = torch.from_numpy(z)
output = vae.decode(z.float())

fig2, ax2 = plt.subplots(nrows=2, ncols=5,figsize=(10, 10), \
                         sharey=True, sharex=True)
j = 0
for fig_x in ax2.flatten():
  im_val = torch.reshape(output[j], (28, 28))
  img_label = "Output"
  j += 1
  plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)
#<выведите тут полученные картинки>
```



Latent Representation

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 лиц в пространстве.

Плюс давайте сделаем такую вещь: у вас есть файл с атрибутами lwf_deepfinetuned.txt, который скачался вместе с базой картинок. Там для каждой картинки описаны атрибуты картинки (имя человека, его пол, цвет кожи и т.п.). Когда будете визуализировать точки латентного пространства на картинке, возьмите какой-нибудь атрибут и покрасьте точки в соответствии со значем атрибута, соответствующего этой точке.

Например, возьмем атрибут "пол". Давайте покрасим точки, которые соответствуют картинкам женщин, в один цвет, а точки, которые соответствуют картинкам мужчин -- в другой.

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 с разными атрибутами.

```
data_t = torch.from_numpy(data[0:3000])
y = data_t[0]
y
```

```
lat = []
male_l = []
female_l = []
sex = attrs['Male']
for i in range(3000):
  y_lat = data_t[i]

  # y_true = torch.from_numpy(y_true)
  # y_true = y_true.to(device)# view(-1, 6075).to(device)
```

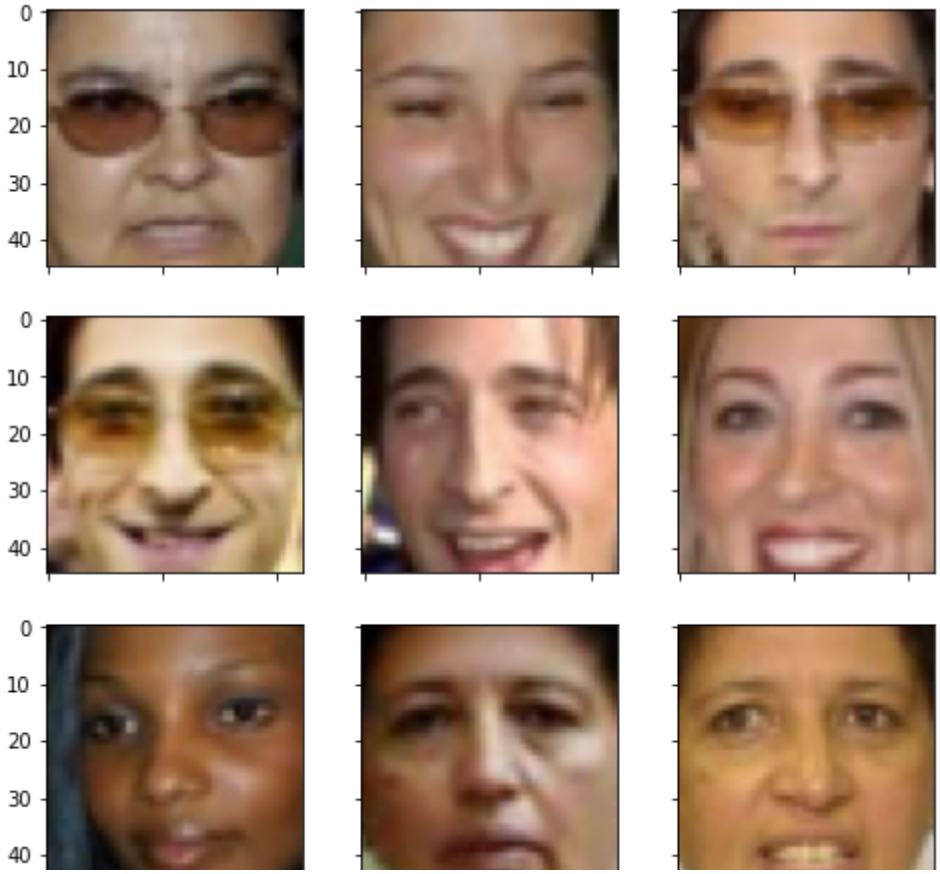
```
    # y_true = y_true.to(device)#.view(-1, 6075).to(device)
    if (sex[i] < -0.5):
      female_l.append(y_lat)
    if (sex[i] > 0.5):
      male_l.append(y_lat)

    y_lat = y_lat.view(-1, 6075).to(device)
    y_pred, y_latent = autoencoder.forward(y_lat.float())
    lat.append(y_latent)


attrs.head(5)
```

	Male	Asian	White	Black	Baby	Child	Youth	Middle Aged
0	1.56835	-1.88904	1.7372	-0.929729	-1.4718	-0.19558	-0.835609	-0.351468
1	0.169851	-0.982408	0.422709	-1.28218	-1.36006	-0.867002	-0.452293	-0.197521
2	0.997749	-1.36419	-0.157377	-0.756447	-1.89183	-0.871526	-0.862893	0.0314447
3	1.12272	-1.9978	1.91614	-2.51421	-2.58007	-1.40424	0.0575511	0.000195882
4	1.07821	-2.0081	1.67621	-2.27806	-2.65185	-1.34841	0.649089	0.0176564

```
a = attrs['Male']
a
```

```
0          1.56835
1         0.169851
2         0.997749
3          1.12272
4          1.07821
           ...
13138    -0.205363
13139      1.95747
13140   -0.0370013
13141     0.282219
13142    0.0711971
Name: Male, Length: 13143, dtype: object
```

```
fig, ax = plt.subplots(nrows=3, ncols=3,figsize=(8, 8), \
                       sharey=True, sharex=True)
j = 0
for fig_x in ax.flatten():
    #random_characters =
    im_val= female_l[j]
    img_label = " "
    j += 1
    plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)
```

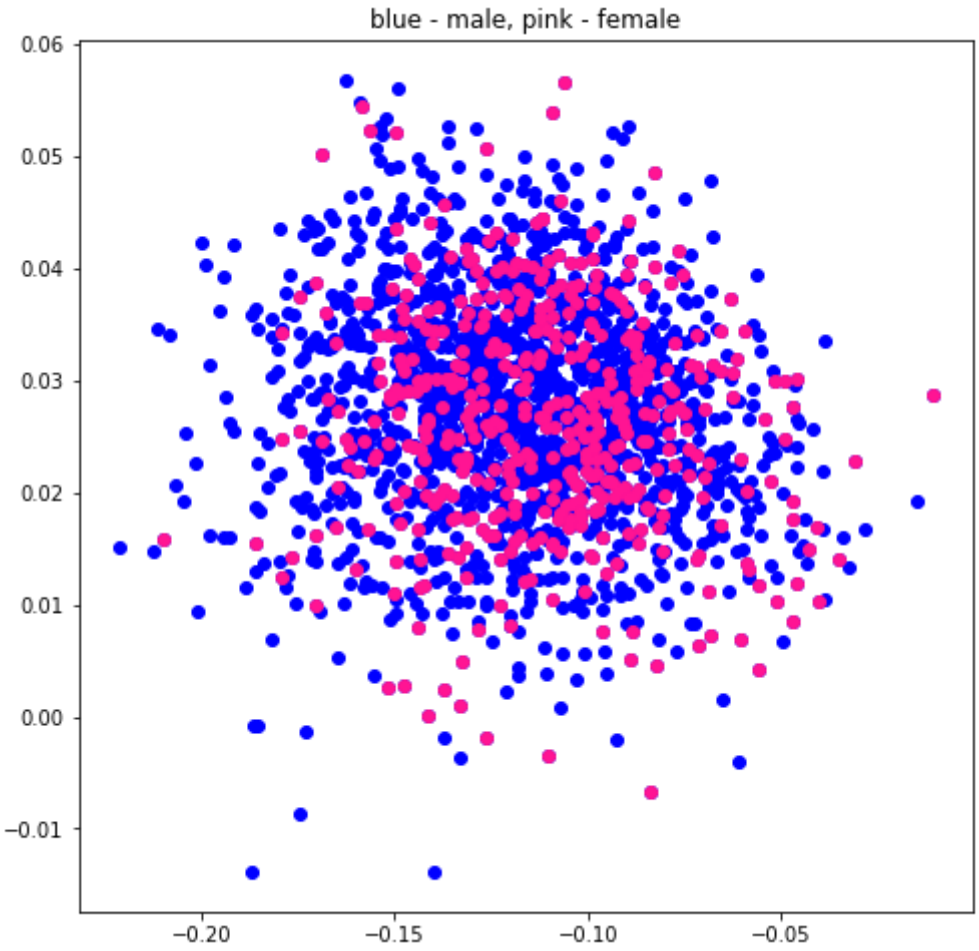



```
latents_male = []
for y_true in male_l:
  true_face.append(y_true)
  y_true = y_true.view(-1, 6075).to(device)
  y_pred, y_latent = autoencoder.forward(y_true.float())
  y_pred = torch.reshape(y_pred, (45, 45, 3))

  #print(y_pred.shape)
  latents_male.append(y_latent)


latents_female = []
for y_true in female_l:
  true_face.append(y_true)
  y_true = y_true.view(-1, 6075).to(device)
  y_pred, y_latent = autoencoder.forward(y_true.float())
  y_pred = torch.reshape(y_pred, (45, 45, 3))
  latents_female.append(y_latent)


from sklearn.manifold import TSNE
np_latents_male = []
np_latents_female = []
for i in range(len(latents_male)):
  a = latents_male[i].detach().numpy()
  np_latents_male.append(a.reshape(100))
np_latents_male = np.array(np_latents_male)
#d_lat_male = TSNE().fit_transform(np_latents_male)

for i in range(len(latents_female)):
  a = latents_male[i].detach().numpy()
```

```
  np_latents_female.append(a.reshape(100))
np_latents_female = np.array(np_latents_female)
#d_lat_all = TSNE().fit_transform(np_latents_female)

np_all = np.concatenate((np_latents_male, np_latents_female), axis=0)
d_lat_all = TSNE().fit_transform(np_all)
#<ваш код получения латентных представлений, применения TSNE и визуализации>


np_latents_male.shape
```

```
(1912, 100)
```

```
fig, ax = plt.subplots()

ax.scatter(np_all[0:1912, 0], np_all[0:1912, 1],
           c = 'blue')    #  цвет точек
ax.scatter(np_all[1913:,0], np_all[1913:,1],
           c = 'deeppink')    #  цвет точек

ax.set_facecolor('white')     #  цвет области Axes
ax.set_title('blue - male, pink - female')     #  заголовок для Axes

fig.set_figwidth(8)     #  ширина и
fig.set_figheight(8)    #  высота "Figure"

plt.show()
```



Что вы думаете о виде латентного представления?

Congrats v2.0!

Conditional VAE (2 балла)

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 мне понадобилось сгенерировать цифру 8. И я подставляю разные варианты шума, и все никак не генерится восьмерка -- у меня получаются то пятерки, то тройки, то четверки. Гадость(

Хотелось бы добавить к нашему AE функцию "выдай мне пожалуйста рандомное число из вот этого вот класса", где классов десять (цифры от 0 до 9 образуют десять классов). Типа я такая говорю "выдай мне случайную восьмерку" и оно генерит случайную восьмерку!

Conditional AE -- так называется вид автоэнкодера, который предоставляет такую возможность. Ну, название "conditional" уже говорит само за себя.

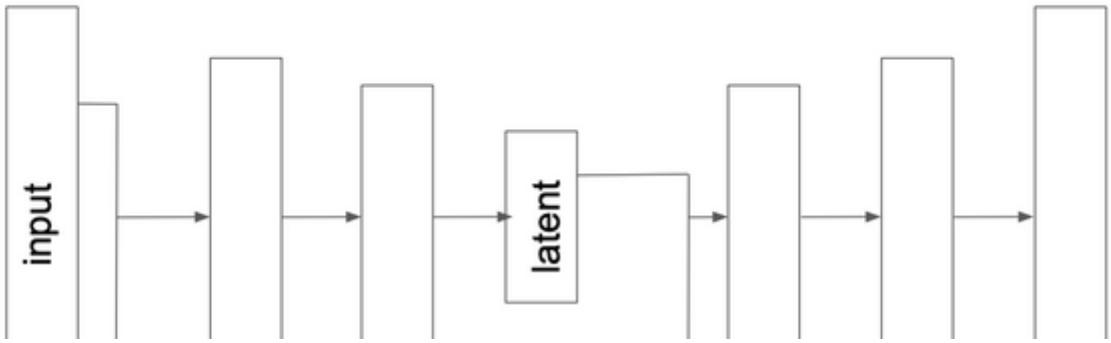
И в этой части проекта мы научимся такие обучать.

Архитектура

На картинке ниже представлена архитектура простого Conditional AE.

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 есть конкатенация картинки и информации о классе (например, вектора из девяти нулей и одной единицы). Первый слой декодера есть конкатенация латентного вектора и информации о классе.



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

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

P.S.

Можно ередавать 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.

Датасет

Здесь я предлагаю вам два вариана. Один попроще, другой -- посложнее, но поинтереснее =)

1. Использовать датасет MNIST (http://yann.lecun.com/exdb/mnist/). Обучать conditional VAE на этом датасете, condition -- класс цифры.
2. Использовать датасет лиц, с которым мы игрались выше. Condition -- пол/раса/улыбки/whatever из lfw_deepfinetuned.txt.

Почему второй вариант "посложнее" -- потому что я сама еще не знаю, получится ли такой CVAE с лицами или нет =) Вы -- исследователи! (не ну это же проект, так и должно быть)

```
def one_hot(x, max_x):
    return torch.eye(max_x + 1)[x]

class CVAE(nn.Module):
    def __init__(self, input_size, max_label, hidden_size=100):
        super(CVAE, self).__init__()

        self.max_label = max_label
        input_size_with_label = input_size + self.max_label + 1
        hidden_size += self.max_label + 1
```

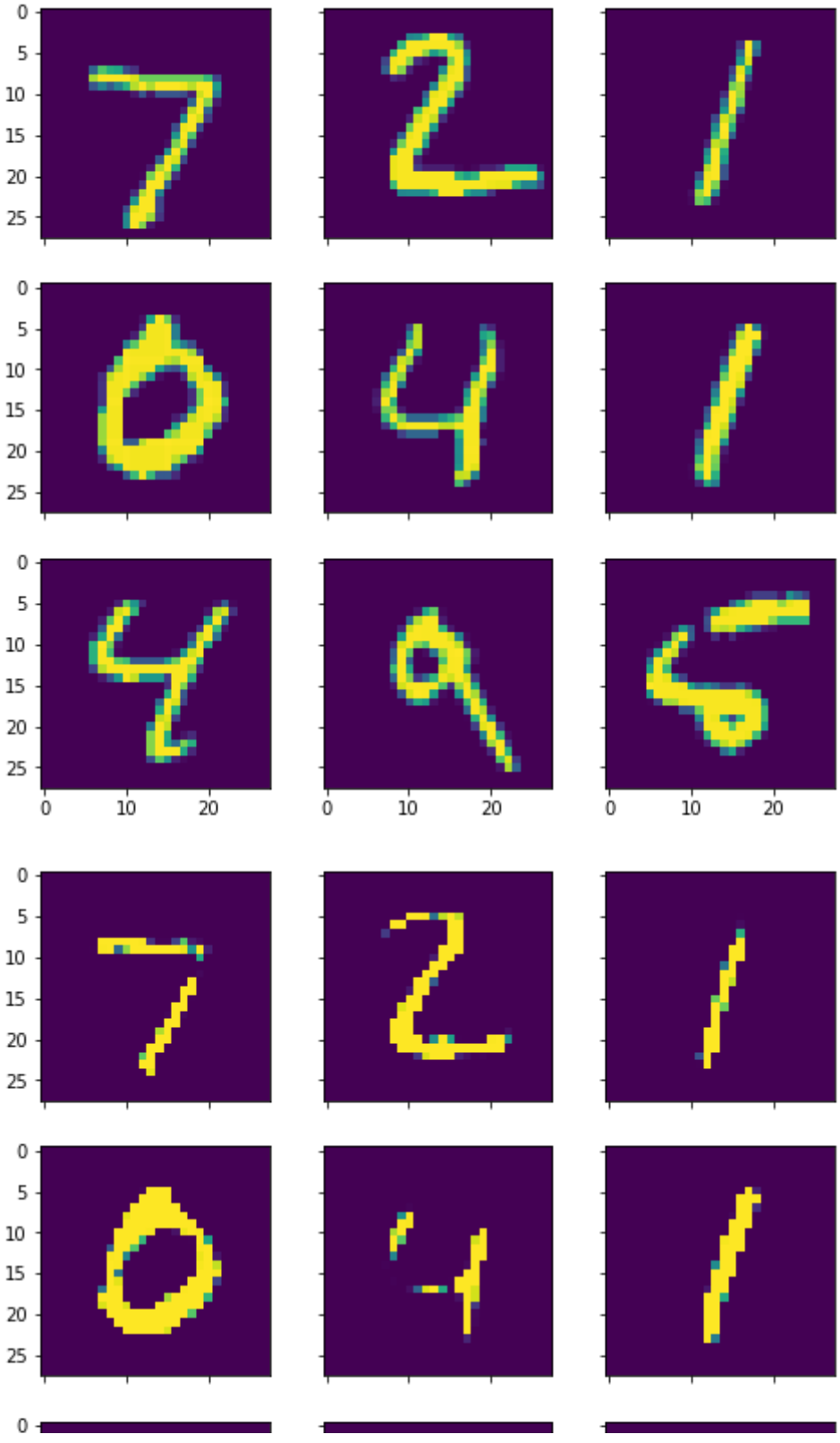
```
            self.fc1 = nn.Linear(input_size_with_label,512)
            self.fc21 = nn.Linear(512, hidden_size)
            self.fc22 = nn.Linear(512, hidden_size)

            self.relu = nn.ReLU()

            self.fc3 = nn.Linear(hidden_size, 512)
            self.fc4 = nn.Linear(512, input_size)

        def encode(self, x, targets):
            x = torch.cat((x, targets), 1)
            x = self.relu(self.fc1(x))
            return self.fc21(x), self.fc22(x)

        def decode(self, z, targets):
            torch.cat((z, targets), 1)
            z = self.fc3(z)
            z = self.relu(z)
            return torch.sigmoid(self.fc4(z))

        def gaussian_sampler(self, mu, logsigma):
            std = torch.exp(0.5 *logsigma)
            eps = torch.randn_like(std)
            return eps.mul(std).add_(mu)

        def forward(self, x, targets):
            targets = one_hot(targets, self.max_label).float()
            mu, logsigma = self.encode(x, targets)
            z = self.gaussian_sampler(mu, logsigma)
            x = self.decode(z, targets)
            return x, mu, logsigma
    #<тут ваш код объявления CVAE, лосса, оптимизатора и тренировки>


    def train_cvae(net, dataloader, epochs=10):
        optim = torch.optim.Adam(net.parameters())
        epoch = 1
        history = []
        for i in range(epochs):
            for y_true, labels in dataloader:
                y_true = y_true.to(device)
                labels = labels.to(device)

                y_true = y_true.view(y_true.size(0), 28*28)

                optim.zero_grad()
                x, mu, logsigma = net(y_true, labels)
                #(x, mu, logsigma, reconstruction):
                loss = loss_vae(y_true, mu, logsigma, x[:, :784])
                loss.backward()
                optim.step()
                train_losses = loss.item()
                #history.append(loss.item())
            print('Epoch: {} \tTraining Loss: {:.6f}'.format(epoch, train_losses))
            history.append(train_losses)
```

```
            epocn += 1

        return history


    cvae = CVAE(28*28, 10)

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


    cvae_history = train_cvae(cvae, train_dataloader_MNIST, 10)
```

```
     Epoch: 1        Training Loss: -3517344.250000
     Epoch: 2        Training Loss: -4681358.000000
     Epoch: 3        Training Loss: -4844251.000000
     Epoch: 4        Training Loss: -4985989.500000
     Epoch: 5        Training Loss: -5044322.000000
     Epoch: 6        Training Loss: -5353391.000000
     Epoch: 7        Training Loss: -5336657.500000
     Epoch: 8        Training Loss: -5253027.500000
     Epoch: 9        Training Loss: -5103573.000000
     Epoch: 10       Training Loss: -5303205.000000
```

```
    train_loss = cvae_history
    plt.figure(figsize=(15,10))
    plt.plot(train_loss, label='Train loss')
    plt.legend(loc='best')
    plt.xlabel("epochs")
    plt.ylabel("loss")
    plt.plot();
```


```
rec_imgs_MHIST = []
#latents_MHIST = []
true_face_MHIST = []
i = 0
x = 9
for y_true, labels in test_dataloader_MNIST:
  if i == x:
    break
  true_face_MHIST.append(y_true)
  y_true = y_true.view(y_true.size(0), 28*28).to(device)
  reconstruction, mu, logsigma = cvae(y_true, labels)
  #mu, logsigma, reconstruction = cvae.forward(y_true.float())
  reconstruction = torch.reshape(reconstruction, (28, 28))

  #print(y_pred.shape)

  rec_imgs_MHIST.append(reconstruction)
  #latents.append(y_latent)
  i += 1

fig0, ax0 = plt.subplots(nrows=3, ncols=3,figsize=(8, 8), \
                         sharey=True, sharex=True)
i = 0
for fig_x in ax0.flatten():
  im_val= torch.reshape(true_face_MHIST[i], (28, 28))
  img_label = " "
  i += 1
  plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)

fig, ax = plt.subplots(nrows=3, ncols=3,figsize=(8, 8), \
                         sharey=True, sharex=True)
i = 0
for fig_x in ax.flatten():
  im_val= rec_imgs_MHIST[i]
  img_label = " "
  i += 1
  plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)
#< тут Ваш код: выведите первые X картинок и их реконструкций из val выборки на экран>
```

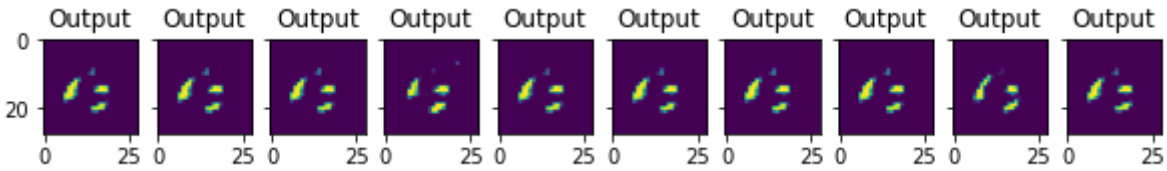


Sampling

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 картинки цифры 5 и 7, а для лиц людей --

восстановить лицо улыбающегося и хмурого человека или лица людей разного пола

```
a = []
z = np.array([np.random.normal(0, 1, 100) for i in range(1)])
#z = torch.from_numpy(z)

for i in range(10):
  a.append(z[0])

a = np.array(a)
a = torch.from_numpy(a)

labels = np.array([0, 1, 2, 3, 4, 5, 6 ,7 ,8, 9])
labels = torch.from_numpy(labels)
labels = one_hot(labels, 10).float()
z = torch.cat((a, labels), 1)
output = cvae.decode(z.float(), labels)

fig2, ax2 = plt.subplots(nrows=1, ncols=10,figsize=(10, 10), \
                         sharey=True, sharex=True)
j = 0
for fig_x in ax2.flatten():
  im_val = torch.reshape(output[j], (28, 28))
  img_label = "Output"
  j += 1
  plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)
#<тут нужно научиться сэмплировать из декодера цифры определенного класса>
```



Splendid! Вы великолепны!

Ну круто же, ну?

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.

```
lat_vae0 = []
lat_vae1 = []
lat_vae2 = []
lat_vae3 = []
lat_vae4 = []
lat_vae5 = []
```

```
    lat_vae9 = []
    lat_vae6 = []
    lat_vae7 = []
    lat_vae8 = []
    lat_vae9 = []

    lat_cvae0 = []
    lat_cvae1 = []
    lat_cvae2 = []
    lat_cvae3 = []
    lat_cvae4 = []
    lat_cvae5 = []
    lat_cvae6 = []
    lat_cvae7 = []
    lat_cvae8 = []
    lat_cvae9 = []

    t = 0

    for y_true, labels in test_dataloader_MNIST:
      t = labels[0]
      y_true = y_true.view(y_true.size(0), 28*28).to(device)
      mu1, logsigma = vae.encode(y_true.float())
      # lat1.append(mu)
      # label1.append(labels)

      label = one_hot(labels, 10).float()
      #z = torch.cat((y_true, label), 1)
      mu2, logsigma = cvae.encode(y_true, label)

      mu1 = mu1.detach().numpy()
      mu1 = mu1.reshape(100)

      mu2 = mu2.detach().numpy()
      mu2 = mu2.reshape(111)

      if (labels[0].item() == 0):
        lat_vae0.append(mu1)
        lat_cvae0.append(mu2)
      if (labels[0].item() == 1):
        lat_vae1.append(mu1)
        lat_cvae1.append(mu2)
      if (labels[0].item() == 2):
        lat_vae2.append(mu1)
        lat_cvae2.append(mu2)
      if (labels[0].item() == 3):
        lat_vae3.append(mu1)
        lat_cvae3.append(mu2)
      if (labels[0].item() == 4):
        lat_vae4.append(mu1)
        lat_cvae4.append(mu2)
      if (labels[0].item() == 5):
        lat_vae5.append(mu1)
        lat_cvae5.append(mu2)
      if (labels[0].item() == 6):
        lat_vae6.append(mu1)
```

```
      lat_vae6.append(mu1)
      lat_cvae6.append(mu2)
    if (labels[0].item() == 7):
      lat_vae7.append(mu1)
      lat_cvae7.append(mu2)
    if (labels[0].item() == 8):
      lat_vae8.append(mu1)
      lat_cvae8.append(mu2)
    if (labels[0].item() == 9):
      lat_vae9.append(mu1)
      lat_cvae9.append(mu2)

l = np.array(lat_vae0)
print(l.shape)

vae_all = np.concatenate((np.array(lat_vae0), np.array(lat_vae1), np.array(lat_vae2), np.a
cvae_all = np.concatenate((lat_cvae0, lat_cvae1, lat_cvae2, lat_cvae3, lat_cvae4, lat_cvae

np_vae = TSNE().fit_transform(vae_all)
np_cvae = TSNE().fit_transform(cvae_all)

#<ваш код получения латентных представлений, применения TSNE и визуализации>
```

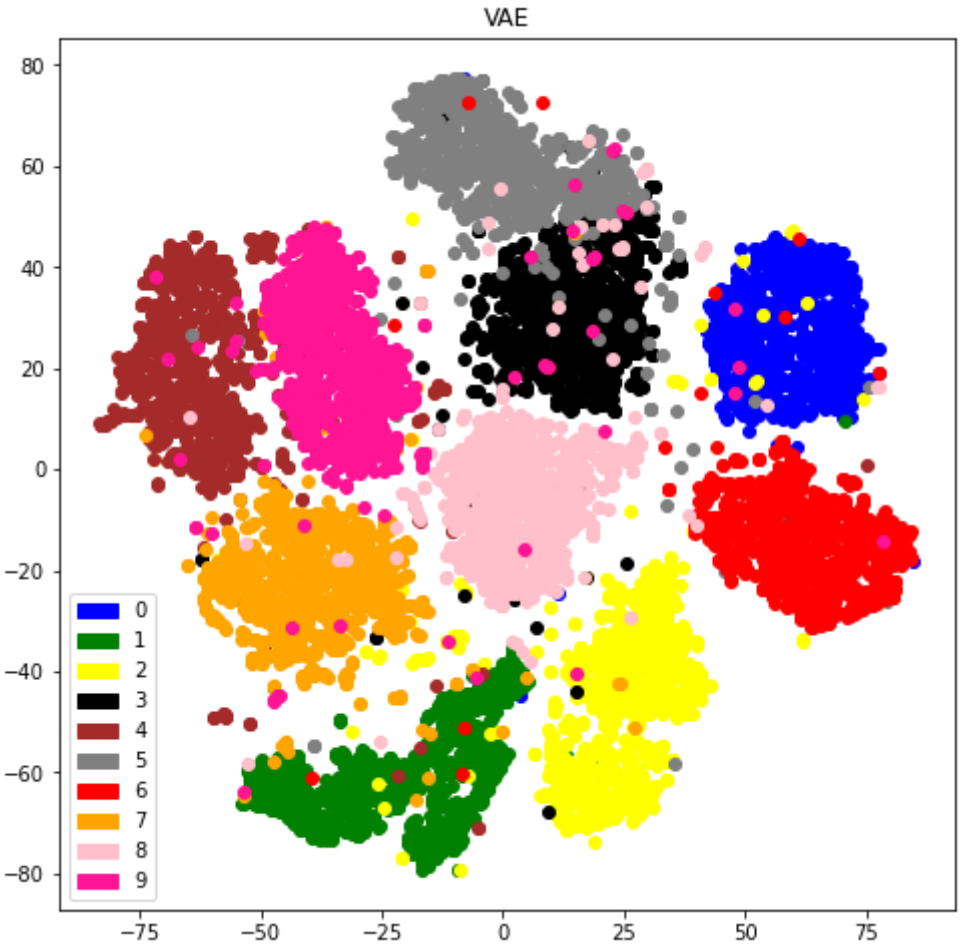
```
(980, 100)
```

```
l = np.array(lat_vae8)
print(l.shape)
```

```
(974, 100)
```

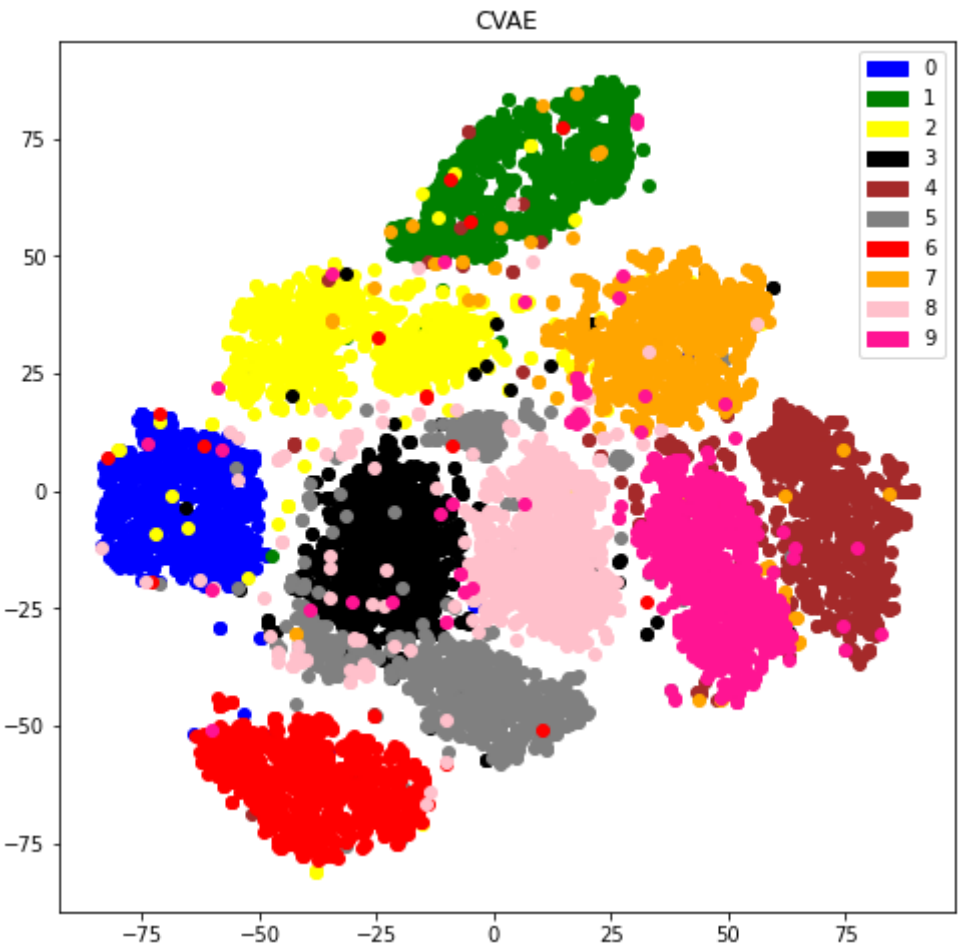
```
#legend((zero, one, two, three, four, five, six, seven, eight, nine), ('zero', 'one', 'two
import matplotlib.patches as mpatches
blue_patch = mpatches.Patch(color='blue', label='0')
green_patch = mpatches.Patch(color='green', label='1')
yellow_patch = mpatches.Patch(color='yellow', label='2')
black_patch = mpatches.Patch(color='black', label='3')
brown_patch = mpatches.Patch(color='brown', label='4')
gray_patch = mpatches.Patch(color='grey', label='5')
red_patch = mpatches.Patch(color='red', label='6')
orange_patch = mpatches.Patch(color='orange', label='7')
pink_patch = mpatches.Patch(color='pink', label='8')
deeppink_patch = mpatches.Patch(color='deeppink', label='9')

fig, ax = plt.subplots()

ax.scatter(np_vae[0:980,0], np_vae[0:980,1],
           c = 'blue')    #  цвет точек
ax.scatter(np_vae[980:2115,0], np_vae[980:2115,1],
           c = 'green')    #  цвет точек
ax.scatter(np_vae[2115:3147,0], np_vae[2115:3147,1],
           c = 'yellow')    #  цвет точек
ax.scatter(np_vae[3147:4157,0], np_vae[3147:4157,1],
           c = 'black')    #  цвет точек
ax.scatter(np_vae[4157:5139,0], np_vae[4157:5139,1],
           c = 'brown')    #  цвет точек
```

```
    ax.scatter(np_vae[5139:6031,0], np_vae[5139:6031,1],
               c = 'grey')    #  цвет точек
    ax.scatter(np_vae[6031:6989,0], np_vae[6031:6989,1],
               c = 'red')    #  цвет точек
    ax.scatter(np_vae[6989:8017,0], np_vae[6989:8017,1],
               c = 'orange')    #  цвет точек
    ax.scatter(np_vae[8017:8991,0], np_vae[8017:8991,1],
               c = 'pink')    #  цвет точек
    ax.scatter(np_vae[8991:,0], np_vae[8991:,1],
               c = 'deeppink')    #  цвет точек
    ax.set_facecolor('white')     #  цвет области Axes
    ax.set_title('VAE')     #  заголовок для Axes
    ax.legend(handles = [blue_patch, green_patch, yellow_patch, black_patch, brown_patch, gray

    fig.set_figwidth(8)     #  ширина и
    fig.set_figheight(8)    #  высота "Figure"

    plt.show()
```




```
    fig1, ax1 = plt.subplots()

    ax1.scatter(np_cvae[0:980,0], np_cvae[0:980,1],
               c = 'blue')    #  цвет точек
    ax1.scatter(np_cvae[980:2115,0], np_cvae[980:2115,1],
               c = 'green')    #  цвет точек
    ax1.scatter(np_cvae[2115:3147,0], np_cvae[2115:3147,1],
               c = 'yellow')    #  цвет точек
    ax1.scatter(np_cvae[3147:4157,0], np_cvae[3147:4157,1],
```

```
            c = 'black')    #  цвет точек
ax1.scatter(np_cvae[4157:5139,0], np_cvae[4157:5139,1],
            c = 'brown')    #  цвет точек
ax1.scatter(np_cvae[5139:6031,0], np_cvae[5139:6031,1],
            c = 'grey')    #  цвет точек
ax1.scatter(np_cvae[6031:6989,0], np_cvae[6031:6989,1],
            c = 'red')    #  цвет точек
ax1.scatter(np_cvae[6989:8017,0], np_cvae[6989:8017,1],
            c = 'orange')    #  цвет точек
ax1.scatter(np_cvae[8017:8991,0], np_cvae[8017:8991,1],
            c = 'pink')    #  цвет точек
ax1.scatter(np_cvae[8991:,0], np_cvae[8991:,1],
            c = 'deeppink')    #  цвет точек
ax1.set_facecolor('white')     #  цвет области Axes
ax1.set_title('CVAE')     #  заголовок для Axes
ax1.legend(handles = [blue_patch, green_patch, yellow_patch, black_patch, brown_patch, gra

fig1.set_figwidth(8)     #  ширина и
fig1.set_figheight(8)    #  высота "Figure"

plt.show()
```



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

BONUS 1: Image Morphing (1 балл)

Предлагаю вам поиграться не только с улыбками, но и с получением из одного человека другого!

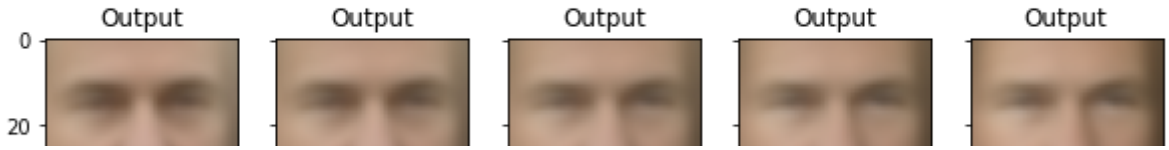
План:

1. Берем две картинки разных людей из датасета
2. Получаем их латентные представления X и Y
3. Складываем латентные представления с коэффициентом α:

   $$\alpha X + (1 - \alpha)Y$$

   где α принимает несколько значений от 0 до 1
4. Визуализируем, как один человек превращается в другого!

```
f = data[0]
s = data[10]
f = torch.from_numpy(f)
s = torch.from_numpy(s)
f = f.view(-1, 6075).to(device)
s = s.view(-1, 6075).to(device)
rec_f = autoencoder.latent(f.float())
rec_s = autoencoder.latent(s.float())

new_rec = []

aplha = 0

for i in range(11):
  new_r = aplha*rec_f + (1 - aplha) * rec_s
  aplha += 0.1
  new_rec.append(autoencoder.reconst(new_r))


fig10, ax10 = plt.subplots(nrows=2, ncols=5,figsize=(10, 10), \
                        sharey=True, sharex=True)
j = 0
for fig_x in ax10.flatten():
  im_val = torch.reshape(new_rec[j], (45, 45, 3))
  img_label = "Output"
  j += 1
  plt_imshow(im_val.data.cpu(), title=img_label, plt_ax=fig_x)
#<тут ваш код>
```



BONUS 2: Denoising (2 балла)

[ ] ↳ *Скрыто 11 ячеек.*

Bonus 3: Image Retrieval (1 балл)

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
  # 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 резул

    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>
```

Bonus 4: Телеграм-бот (3 балла)

Вы можете написать телеграм-бота с функционалом AE. Например, он может добавлять к вашей фотографии улыбку или искать похожие на ваше лицо лица среди лиц датасета.

Код бота должно быть можно проверить!

Эпилог