Школа глубокого обучения ФПМИ МФТИ

Домашнее задание: **Autoencoders, Variational Autoencoders**

Stepik: Павел Цветов**, User ID: 39555013**

In [1]:

```
!nvidia-smi
```

```
Mon May 31 19:17:38 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.19.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   40C    P0    28W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Часть **1. Vanilla Autoencoder (10** баллов)

1.1. Подготовка данных (0.5 балла)

In [2]:

```
import numpy as np
import pandas as pd

import os
import gc

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data as data_utils

from torchvision import datasets
from torchvision import transforms
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.autograd import Variable


import skimage.io
import skimage
from skimage.transform import resize
from sklearn.neighbors import NearestNeighbors
from sklearn.model_selection import train_test_split

from tqdm.notebook import tqdm
from IPython.display import clear_output

import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

plt.style.use('ggplot')

%matplotlib inline
```

In [3]:

```
# def setup_seed(seed):
#    np.random.seed(seed)
#    torch.manual_seed(seed)
#    torch.cuda.manual_seed_all(seed)
#    torch.backends.cudnn.deterministic = True
#    torch.backends.cudnn.benchmark = False

# seed = 41

# setup_seed(seed)
```

In [4]:

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

```
cuda
```

In [5]:

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

    #download if not exists
```

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

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

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


    #read photos
    photo_ids = []
    for dirpath, dirnames, filenames in os.walk(images_name):
        for fname in filenames:
            if fname.endswith(".jpg"):
                fpath = os.path.join(dirpath,fname)
                photo_id = fname[:-4].replace('_',' ').split()
                person_id = ' '.join(photo_id[:-1])
                photo_number = int(photo_id[-1])
                photo_ids.append({'person':person_id,'imagenum':photo_number,'photo_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)\
                                .apply(lambda img:img[dy:-dy,dx:-dx])\
                                .apply(lambda img: resize(img,[dimx,dimy]))

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

    return all_photos, all_attrs
```

In [6]:

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

In [7]:

```
attrs.head()
```

Out[7]:

	Male	Asian	White	Black	Baby	Child	Youth	Middle Aged	Senior	Black Hair	Blond Hair	Brow Ha
0	1.56835	-1.88904	1.7372	-0.929729	-1.4718	-0.19558	-0.835609	-0.351468	-1.01253	-0.719593	-0.632401	0.46483
1	0.169851	-0.982408	0.422709	-1.28218	-1.36006	-0.867002	-0.452293	-0.197521	-0.956073	-0.802107	-0.736883	0.29455

2	0.997749	-1.36419	0.157377	0.756447	1.89183	0.871526	-0.862893	0.0314447	-1.34152	0.0900375	-1.20073	-0.3324
3	Male 1.12272	Asian -1.9978	White 1.91614	Black -2.51421	Baby 2.58007	Child -1.40424	Youth 0.0575511	Middle Aged 0.000195862	Senior -1.27351	Black Hair -1.43146	Blond Hair 0.0705188	Brow Ha 0.33923
4	1.07821	-2.0081	1.67621	-2.27806	-2.65185	-1.34841	0.649089	0.0176564	-1.88911	-1.85721	-0.568057	0.84037

In [8]:

```
data = np.array(data, dtype='float32') # convert images to float32 type

dataset = torch.from_numpy(data)
dataset = dataset.transpose(1, 3) # reshape tensor dataset to shape Batch x Channels x H
eight x Width
```

In [9]:

```
dataset.shape
```

Out[9]:

```
torch.Size([13143, 3, 64, 64])
```

In [10]:

```
fig, axes = plt.subplots(2, 5, figsize=(16,8))

for ax in axes.flat:
    i = int(np.random.uniform(0, len(data) - 1))
    ax.imshow(dataset[i].transpose(0, 2).numpy())
    ax.axis('off')
```

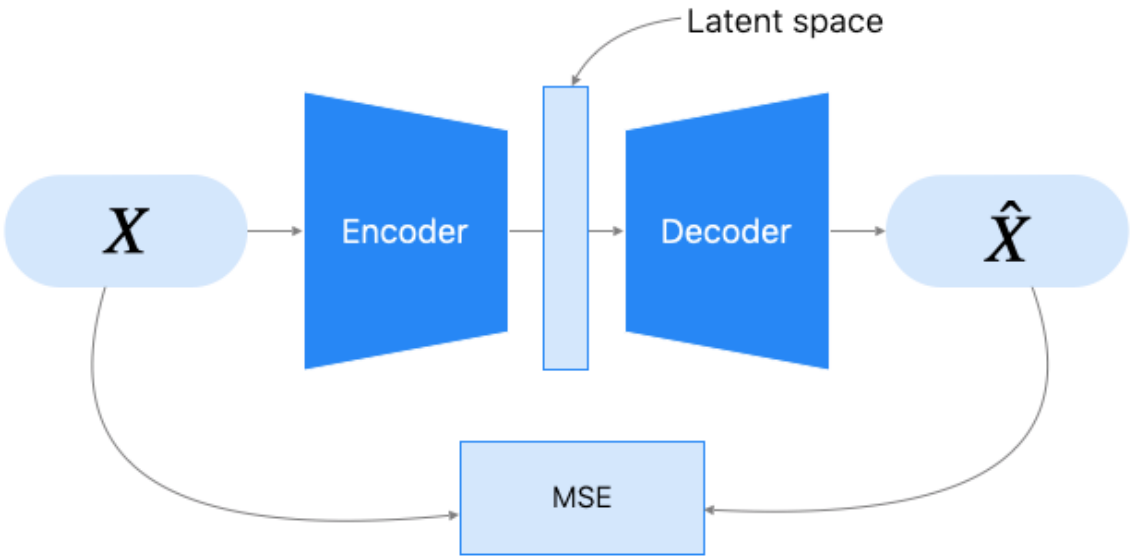
Разбейте выборку картинок на **train** и **val,** выведите несколько картинок в **output,** чтобы посмотреть, как они выглядят, и приведите картинки к тензорам **pytorch,** чтобы можно было скормить их сети:

In [11]:

```
train_photos, test_photos = train_test_split(dataset, test_size=0.1)

train_dataloader = DataLoader(train_photos, batch_size=128, shuffle=True)
test_dataloader = DataLoader(test_photos, batch_size=128, shuffle=False)
```

1.2. Архитектура модели (1.5 балла)

В этом разделе мы напишем и обучем обычный автоэнкодер.



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

In [12]:

```
from torchsummary import summary
```

In [13]:

```
class ConvAutoencoder(nn.Module):
    def __init__(self, hidden):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=2, padding=1
),
            nn.ReLU(),
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=2, padding=
1),
            nn.ReLU(),
            nn.Conv2d(in_channels=32, out_channels=128, kernel_size=5),
            nn.Flatten(),
            nn.Linear(18432, 4096),
            nn.ReLU(),
            nn.Linear(4096, 2048),
            nn.ReLU(),
            nn.Linear(2048, hidden)
        )
        self.decoder = nn.Sequential(
            nn.Linear(hidden, 2048),
            nn.ReLU(),
            nn.Linear(2048, 4096),
            nn.ReLU(),
            nn.Linear(4096, 18432),
            nn.Unflatten(1, (128, 12, 12)),
            nn.ConvTranspose2d(128, 32, 5),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, 3, 2, 1, output_padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(16, 3, 3, 2, 1, output_padding=1),
            nn.Sigmoid()
        )

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

    def decode(self, z):
```

```
        return self.decoder(z)

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

In [14]:

```
class LinearAutoencoder(nn.Module):
    def __init__(self, input_size, hidden):
      super().__init__()
      self.encoder = nn.Sequential(
          nn.Linear(input_size, 512),
          nn.ReLU(),
          nn.Linear(512, 256),
          nn.ReLU(),
          nn.Linear(256, 128),
          nn.ReLU(),
          nn.Linear(128, 64),
          nn.ReLU(),
          nn.Linear(64, hidden)
      )

      self.decoder = nn.Sequential(
          nn.Linear(hidden, 64),
          nn.ReLU(),
          nn.Linear(64, 128),
          nn.ReLU(),
          nn.Linear(128, 256),
          nn.ReLU(),
          nn.Linear(256, 512),
          nn.ReLU(),
          nn.Linear(512, input_size)
      )

    def forward(self, x):
      shapes = x.shape
      x = x.view(x.size(0), -1)
      return self.decoder(self.encoder(x)).view(*shapes)

    def encode(self, x):
      x = x.view(x.size(0), -1)
      return self.encoder(x)

    def decode(self, x):
      x = x.view(x.size(0), -1)
      return self.decoder(x)
```

1.3 Обучение (2 балла)

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

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

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

In [15]:

```
gc.collect()
torch.cuda.empty_cache()
```

Linear Autoencoder

In [16]:

```
def train(model, optimizer, loss_fn, epochs, train_dataloader, test_dataloader, schedule
r=None):

  x_test = next(iter(test_dataloader))
  train_history, test_history = [], []

  for epoch in tqdm(range(epochs)):

      model.train()
      train_loss = 0

      for x in train_dataloader:

          optimizer.zero_grad()

          x = x.to(device)
          x_pred = model(x)

          loss = loss_fn(x_pred, x)
          loss.backward()

          optimizer.step()

          if scheduler is not None:
            scheduler.step()

          train_loss += loss.item()

      train_loss /= len(train_dataloader)

      train_history.append(train_loss)

      model.eval()

      x = x_test.to(device)

      x_pred = model(x)
      loss = loss_fn(x_pred, x)

      test_loss = loss.item()

      test_history.append(test_loss)

      x = x.cpu().data
      x_pred = x_pred.cpu().data

      clear_output(wait=True)

      fig, axes = plt.subplots(2, 10, figsize=(16,8))
      fig.suptitle(f'{epoch+1} / {epochs} - train_loss: {train_loss:.4f}, test_loss: {te
st_loss:.4f}')

      for ax in axes.flat:
          ax.set_xticks([]); ax.set_yticks([])
      for i, ax in enumerate(axes[0]):
          ax.imshow(x[i].transpose(0, 2).numpy())
          ax.set_title('Real')
      for i, ax in enumerate(axes[1]):
          ax.imshow(x_pred[i].transpose(0, 2).numpy())
          ax.set_title('Generated')

      plt.show()

  return train_history, test_history
```

In [17]:

```
epochs = 50
```

In [18]:

```
loss_fn = nn.MSELoss()

model = LinearAutoencoder(3*64*64, 16)
model = model.to(device)

optimizer = optim.AdamW(model.parameters(), lr=1e-3)
```

In [19]:

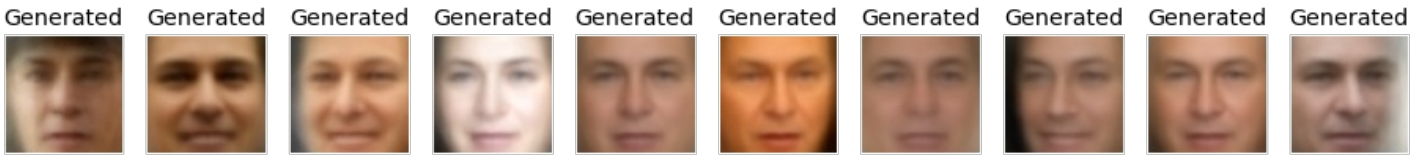
```
summary(model, input_size=(3, 64, 64))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 512]       6,291,968
              ReLU-2                  [-1, 512]               0
            Linear-3                  [-1, 256]         131,328
              ReLU-4                  [-1, 256]               0
            Linear-5                  [-1, 128]          32,896
              ReLU-6                  [-1, 128]               0
            Linear-7                   [-1, 64]           8,256
              ReLU-8                   [-1, 64]               0
            Linear-9                   [-1, 16]           1,040
           Linear-10                   [-1, 64]           1,088
             ReLU-11                   [-1, 64]               0
           Linear-12                  [-1, 128]           8,320
             ReLU-13                  [-1, 128]               0
           Linear-14                  [-1, 256]          33,024
             ReLU-15                  [-1, 256]               0
           Linear-16                  [-1, 512]         131,584
             ReLU-17                  [-1, 512]               0
           Linear-18                [-1, 12288]       6,303,744
================================================================
Total params: 12,943,248
Trainable params: 12,943,248
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 0.12
Params size (MB): 49.37
Estimated Total Size (MB): 49.54
----------------------------------------------------------------
```

In [20]:

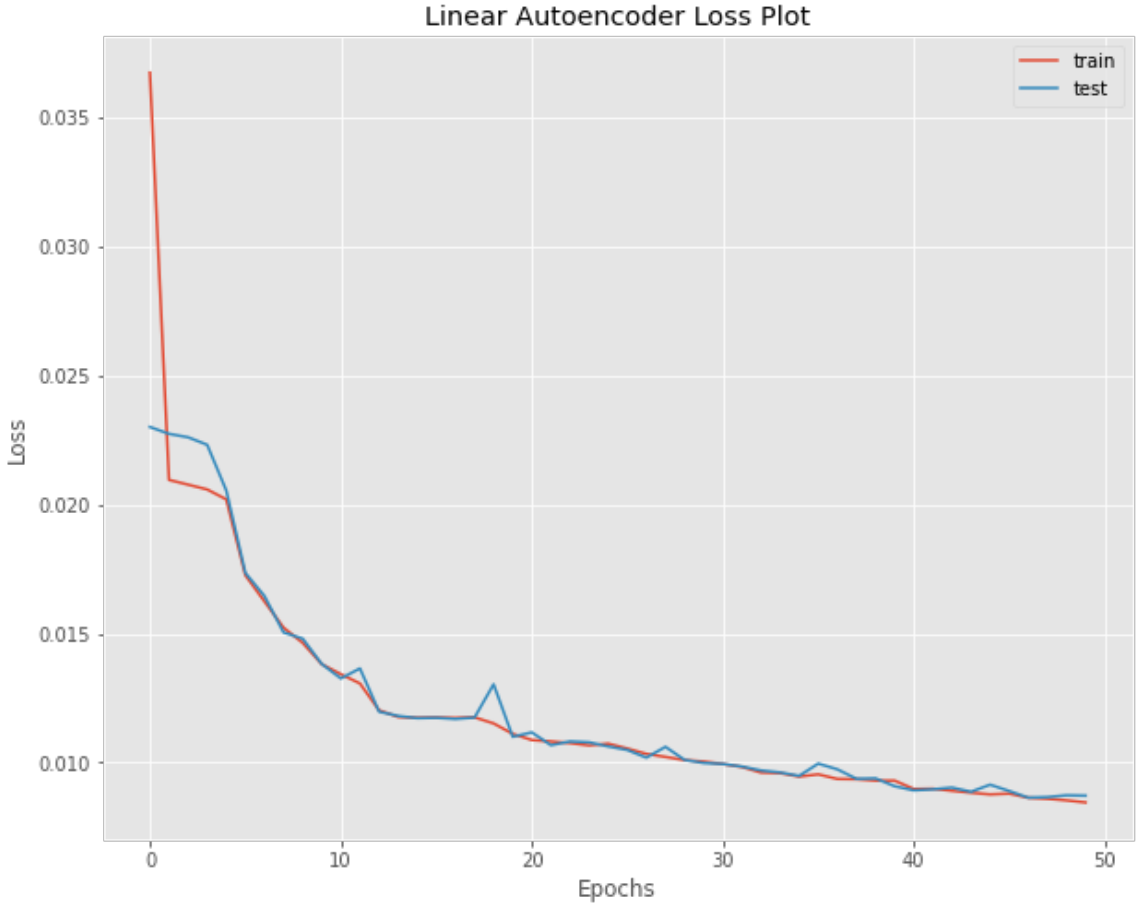
```
train_history, test_history = train(model, optimizer, loss_fn, epochs, train_dataloader,
test_dataloader)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0.
.255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0.
.255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0.
.255] for integers).
```



In [21]:

```
plt.figure(figsize=(10, 8))

plt.plot(range(epochs), train_history, label='train')
plt.plot(range(epochs), test_history, label='test')

plt.title('Linear Autoencoder Loss Plot')
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.legend()
plt.show()
```



Convolutional Autoencoder

In [22]:

```
gc.collect()
torch.cuda.empty_cache()
```

In [23]:

```
epochs = 20
```

In [24]:

```
model = ConvAutoencoder(16)
model = model.to(device)

optimizer = optim.AdamW(model.parameters(), lr=1e-3)
```

In [25]:

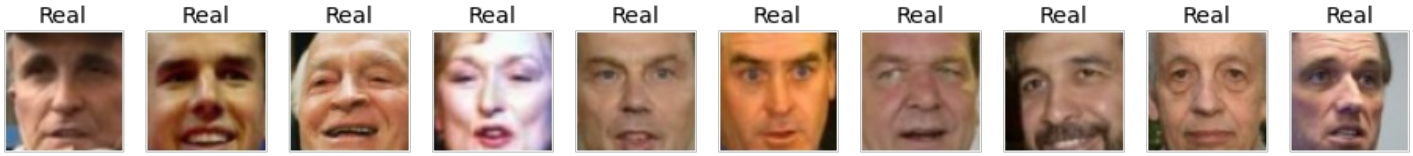
```
summary(model, input_size=(3, 64, 64))
```

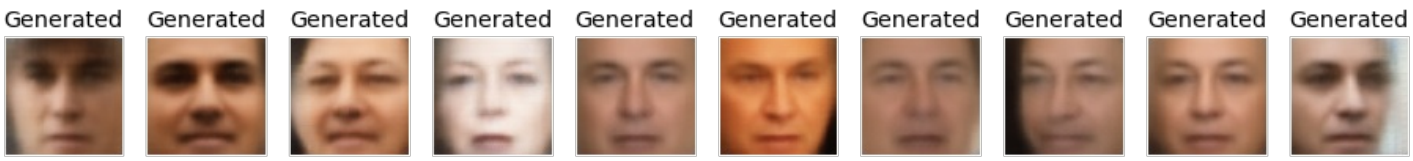
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 32, 32]             448
              ReLU-2           [-1, 16, 32, 32]               0
            Conv2d-3           [-1, 32, 16, 16]           4,640
              ReLU-4           [-1, 32, 16, 16]               0
            Conv2d-5          [-1, 128, 12, 12]         102,528
           Flatten-6                [-1, 18432]               0
            Linear-7                 [-1, 4096]      75,501,568
              ReLU-8                 [-1, 4096]               0
            Linear-9                 [-1, 2048]       8,390,656
             ReLU-10                 [-1, 2048]               0
           Linear-11                   [-1, 16]          32,784
           Linear-12                 [-1, 2048]          34,816
             ReLU-13                 [-1, 2048]               0
           Linear-14                 [-1, 4096]       8,392,704
             ReLU-15                 [-1, 4096]               0
           Linear-16                [-1, 18432]      75,515,904
        Unflatten-17          [-1, 128, 12, 12]               0
  ConvTranspose2d-18           [-1, 32, 16, 16]         102,432
             ReLU-19           [-1, 32, 16, 16]               0
  ConvTranspose2d-20           [-1, 16, 32, 32]           4,624
             ReLU-21           [-1, 16, 32, 32]               0
  ConvTranspose2d-22            [-1, 3, 64, 64]             435
          Sigmoid-23            [-1, 3, 64, 64]               0
================================================================
Total params: 168,083,539
Trainable params: 168,083,539
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 1.69
Params size (MB): 641.19
Estimated Total Size (MB): 642.92
----------------------------------------------------------------
```

In [26]:

```
train_history, test_history = train(model, optimizer, loss_fn, epochs, train_dataloader,
test_dataloader)
```

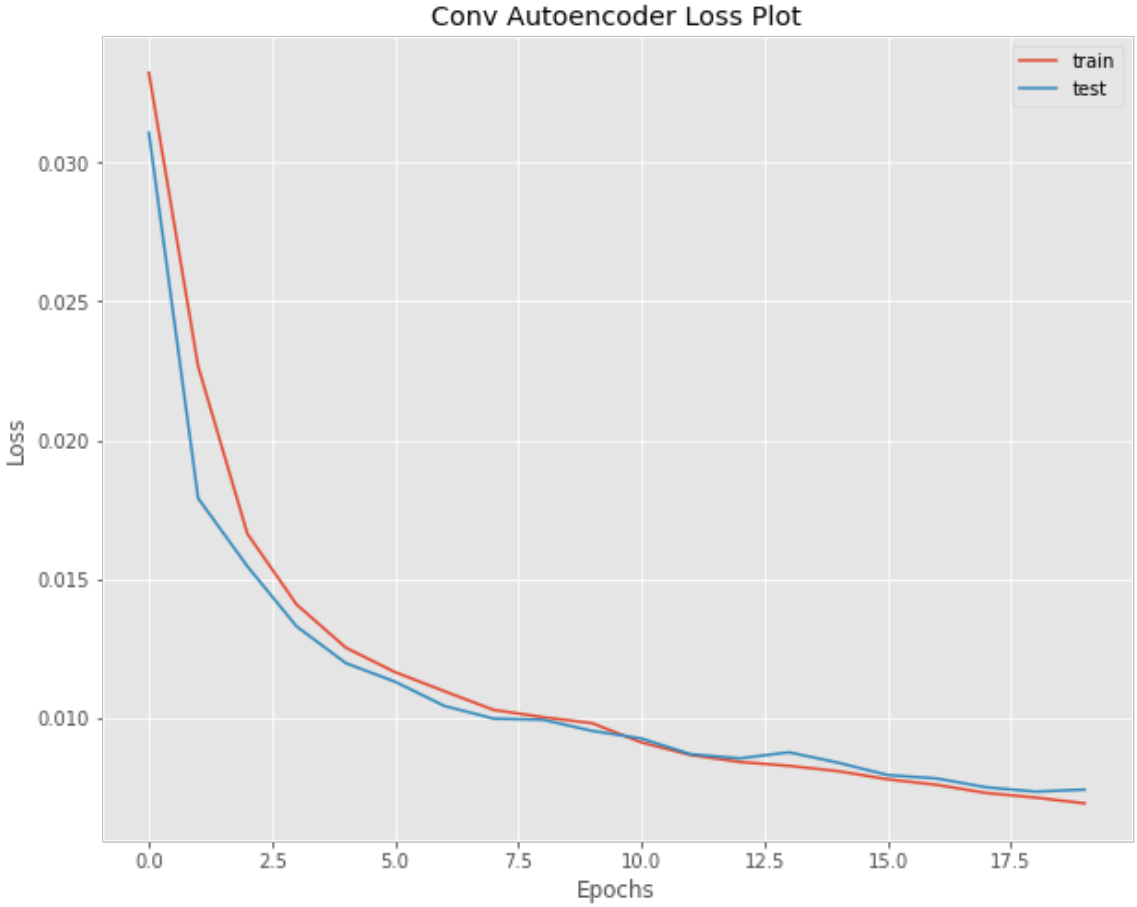
```
20 / 20 - train_loss: 0.0069, test_loss: 0.0074
```





In [27]:

```
plt.figure(figsize=(10, 8))

plt.plot(range(epochs), train_history, label='train')
plt.plot(range(epochs), test_history, label='test')

plt.title('Conv Autoencoder Loss Plot')
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.legend()
plt.show()
```



1.4. Sampling (2 балла)

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

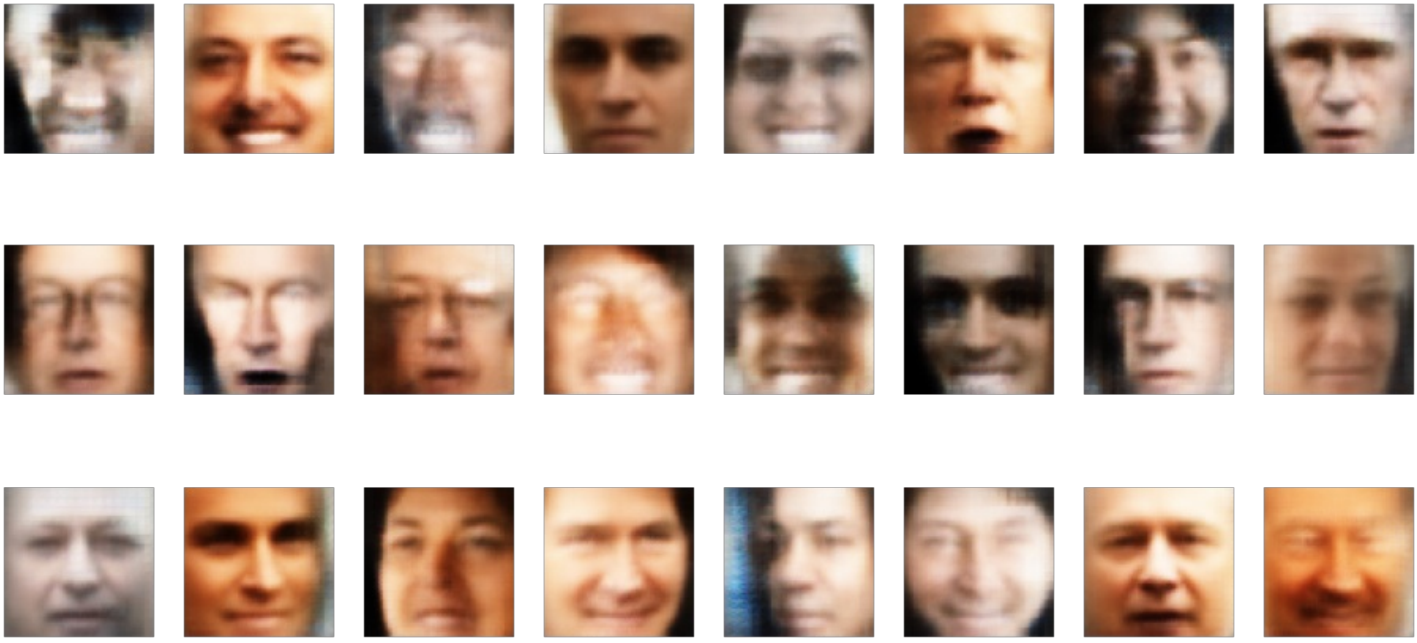
Давайте возьмем и подсунем декодеру какие-нибудь сгенерированные нами векторы (например, из нормального распределения) и посмотрим на результат реконструкции декодера:

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

In [28]:

```
std_mu, std_sigma = 0, 1
z = std_mu + std_sigma * np.random.randn(24, 16)
z = torch.from_numpy(z.astype('float32')).to(device)
output = model.decode(z).detach().cpu().data

fig, axes = plt.subplots(3, 8, figsize=(16,8))
for i, ax in enumerate(axes.flat):
    ax.imshow(output[i].transpose(0,2).numpy())
```

```
    ax.axis('off')
```



Time to make fun! (4 балла)

Давайте научимся пририсовывать людям улыбки =)

План такой:

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

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

1. Вычислить латентный вектор для всех улыбающихся людей (прогнать их через **encoder**) и то же для всех грустненьких
2. Вычислить, собственно, вектор улыбки -- посчитать разность между средним латентным вектором улыбающихся людей и средним латентным вектором грустных людей
3. А теперь приделаем улыбку грустному человеку: добавим полученный в пункте 3 вектор к латентному вектору грустного человека и прогоним полученный вектор через **decoder**. Получим того же человека, но уже не грустненького!

In [29]:

```
attrs.head()
```

Out[29]:

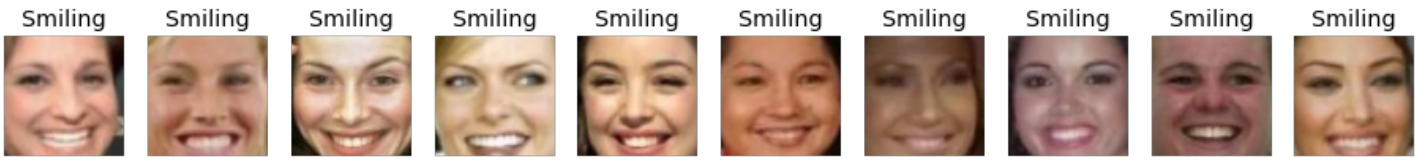
	Male	Asian	White	Black	Baby	Child	Youth	Middle Aged	Senior	Black Hair	Blond Hair	Brow Ha
0	1.56835	-1.88904	1.7372	-0.929729	-1.4718	-0.19558	-0.835609	-0.351468	-1.01253	-0.719593	-0.632401	0.4648
1	0.169851	-0.982408	0.422709	-1.28218	-1.36006	-0.867002	-0.452293	-0.197521	-0.956073	-0.802107	-0.736883	0.2945
2	0.997749	-1.36419	-0.157377	-0.756447	-1.89183	-0.871526	-0.862893	0.0314447	-1.34152	-0.0900375	-1.20073	-0.332
3	1.12272	-1.9978	1.91614	-2.51421	-2.58007	-1.40424	0.0575511	0.000195882	-1.27351	-1.43146	-0.0705188	0.3392
4	1.07821	-2.0081	1.67621	-2.27806	-2.65185	-1.34841	0.649089	0.0176564	-1.88911	-1.85721	-0.568057	0.8403

In [30]:

```
def get_top_faces_by_category(df: pd.DataFrame, attribute: str, n: int=20):

    filtered_df = df.sort_values(by=attribute, ascending=False)[attribute]
    return filtered_df[:n].index.tolist()

def show_faces_by_category(category: str, n: int=20):

    faces = get_top_faces_by_category(attrs, category, n=n)
    fig, axes = plt.subplots(1, 10, figsize=(16,8))

    for ax, im in zip(axes.flat, dataset[faces]):
        ax.imshow(im.transpose(0,2).numpy())
        ax.set_title(category)
        ax.axis('off')

    plt.show()

def transform_faces_by_category(source_category: str, target_category: str, n: int=15):

    source_faces = get_top_faces_by_category(attrs, source_category, n=n)
    target_faces = get_top_faces_by_category(attrs, target_category, n=n)

    source_encoded = model.encode(dataset[source_faces].to(device))
    target_encoded = model.encode(dataset[target_faces].to(device))

    diff_vector = target_encoded.mean(axis=0) - source_encoded.mean(axis=0)

    result = model.decode((source_encoded + diff_vector))
    result.to(device)
    result = result.detach().cpu().data

    source = dataset[source_faces]

    fig, axes = plt.subplots(2, 10, figsize=(16, 8))

    for ax in axes.flat:
      for i, ax in enumerate(axes[0]):
        ax.imshow(source[i].transpose(0,2).numpy())
        ax.axis('off')
        ax.set_title('Original')
      for i, ax in enumerate(axes[1]):
        ax.imshow(result[i].transpose(0,2).numpy())
        ax.axis('off')
        ax.set_title('Transformed')
```

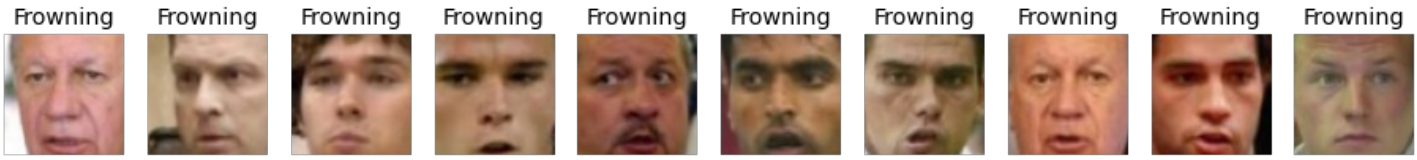
In [31]:

```
show_faces_by_category('Smiling')
```



In [32]:

```
show_faces_by_category('Frowning')
```

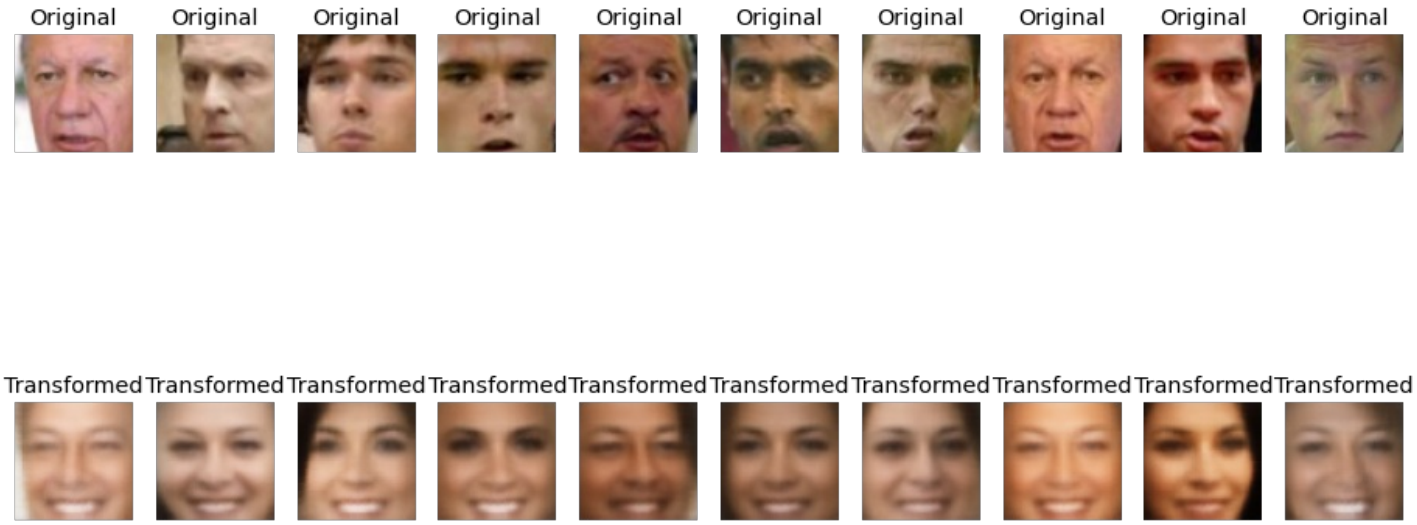


In [33]:

```
show_faces_by_category('Sunglasses')
```

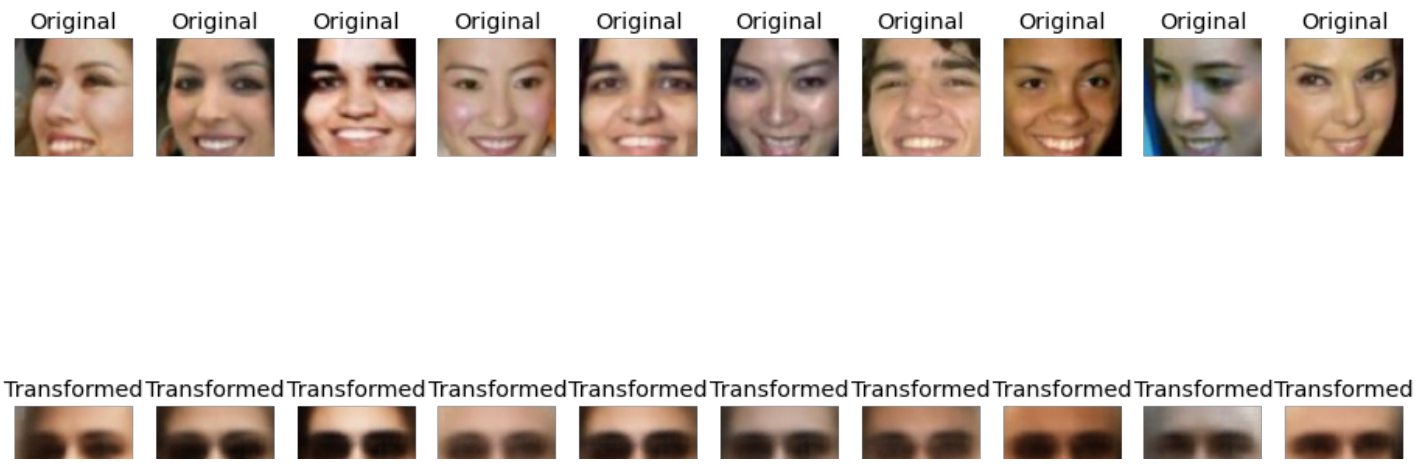
In [34]:

```
transform_faces_by_category('Frowning', 'Smiling', n=30)
```



In [35]:

```
transform_faces_by_category('Youth', 'Sunglasses', n=30)
```



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**

In [36]:

```
batch_size = 64

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

# Data Loader (Input Pipeline)
train_dataloader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_si
ze, shuffle=True)
test_dataloader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_siz
e, shuffle=False)
```

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**!**

In [37]:

```
def OneHotEncoder(idx, n):
    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 Encoder(nn.Module):

    def __init__(self, conditional):

        super().__init__()
        self.conditional = conditional
        s0 = 28*28+10 if conditional else 28*28

        self.encoder = nn.Sequential(
            nn.Linear(s0, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.BatchNorm1d(128),
            nn.ReLU()
        )

        self.linear_means = nn.Linear(128, 16)
        self.linear_log_var = nn.Linear(128, 16)

    def forward(self, x, c):
        if self.conditional:
            c = OneHotEncoder(c, n=10)
            x = torch.cat((x, c), dim=-1)
        x = self.encoder(x)
```

```
        means = self.linear_means(x)
        log_vars = self.linear_log_var(x)
        return means, log_vars

class Decoder(nn.Module):

    def __init__(self, conditional):

        super().__init__()
        self.conditional = conditional
        s0 = 16+10 if conditional else 16

        self.decoder = nn.Sequential(
            nn.Linear(s0, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Linear(512, 28*28),
            nn.Sigmoid()
        )

    def forward(self, z, c):
        if self.conditional:
            c = OneHotEncoder(c, n=10)
            z = torch.cat((z, c), dim=-1)
        x = self.decoder(z)
        return x

class LinearVAE(nn.Module):

    def __init__(self, conditional=False):

        super().__init__()

        self.encoder = Encoder(conditional)
        self.decoder = Decoder(conditional)

    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 encode(self, x, c=None):
        x = x.view(-1, 28*28)
        means, log_var = self.encoder(x, c)
        z = self.reparameterize(means, log_var)
        return means, log_var, z

    def decode(self, z, c=None):
        recon_x = self.decoder(z, c)
        recon_x = recon_x.view(-1, 1, 28, 28)
        return recon_x

    def forward(self, x, c=None):
        means, log_var, z = self.encode(x, c)
        recon_x = self.decode(z, c)
        return recon_x, means, log_var, z
```

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:

Общий лосс будет выглядеть так:

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

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.

In [38]:

```
def loss_vae(reconstructed, x, mu, log_var):

    BCE = torch.nn.functional.binary_cross_entropy(reconstructed.view(-1, 28*28), x.view
(-1, 28*28), reduction='sum')
    KL = -0.5 * torch.sum(1 + log_var - mu.pow(2) - log_var.exp())

    return BCE + KL
```

И обучим модель:

In [39]:

```
gc.collect()
torch.cuda.empty_cache()
```

In [40]:

```
loss_fn = loss_vae

model = LinearVAE()
model.to(device)

optimizer = optim.AdamW(model.parameters(), lr=1e-3, weight_decay=1e-5)
```

In [41]:

```
epochs = 50
```

In [42]:

```
summary(model, input_size=(3, 28, 28))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 512]         401,920
       BatchNorm1d-2                  [-1, 512]           1,024
              ReLU-3                  [-1, 512]               0
            Linear-4                  [-1, 256]         131,328
       BatchNorm1d-5                  [-1, 256]             512
              ReLU-6                  [-1, 256]               0
            Linear-7                  [-1, 128]          32,896
       BatchNorm1d-8                  [-1, 128]             256
              ReLU-9                  [-1, 128]               0
           Linear-10                   [-1, 16]           2,064
           Linear-11                   [-1, 16]           2,064
          Encoder-12       [[-1, 16], [-1, 16]]               0
           Linear-13                  [-1, 256]           4,352
      BatchNorm1d-14                  [-1, 256]             512
             ReLU-15                  [-1, 256]               0
```

```
           Linear-16                  [-1, 512]         131,584
      BatchNorm1d-17                  [-1, 512]           1,024
             ReLU-18                  [-1, 512]               0
           Linear-19                  [-1, 784]         402,192
          Sigmoid-20                  [-1, 784]               0
          Decoder-21                  [-1, 784]               0
================================================================
Total params: 1,111,728
Trainable params: 1,111,728
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.05
Params size (MB): 4.24
Estimated Total Size (MB): 4.30
----------------------------------------------------------------
```

In [43]:

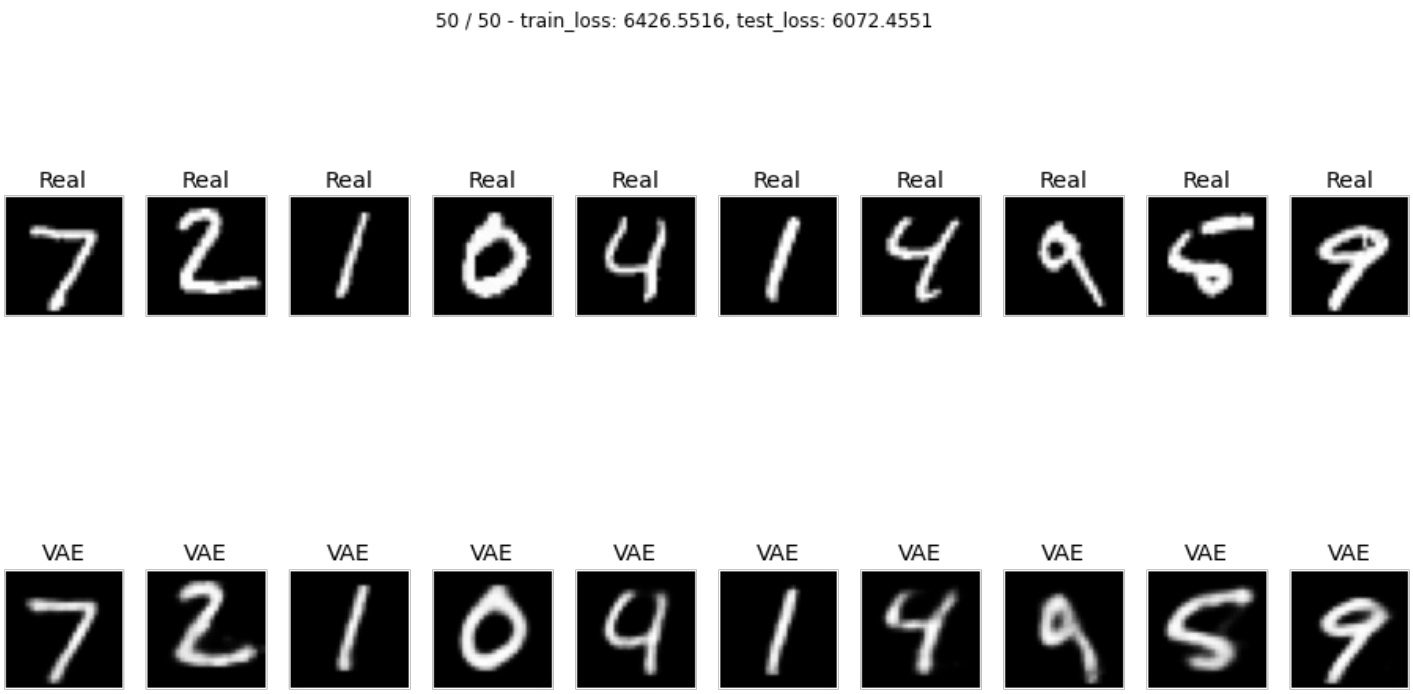
```
def train_vae(model, optimizer, loss_fn, epochs, train_dataloader, test_dataloader, sche
duler=None):

  x_test, y_test = next(iter(test_dataloader))
  train_history, test_history = [], []

  for epoch in tqdm(range(epochs)):

      model.train()
      train_loss = 0

      for x, y in train_dataloader:

          optimizer.zero_grad()

          x = x.to(device)
          y = y.to(device)

          reconstructed, mean, log_var, z = model(x, y)

          loss = loss_fn(reconstructed, x, mean, log_var)
          loss.backward()

          optimizer.step()

          if scheduler is not None:
            scheduler.step()

          train_loss += loss.item()

      train_loss /= len(train_dataloader)

      train_history.append(train_loss)

      model.eval()

      x, y = x_test, y_test

      x = x.to(device)
      y = y.to(device)

      reconstructed, mean, log_var, z = model(x, y)
      loss = loss_fn(reconstructed, x, mean, log_var)
      test_loss = loss.item()

      test_history.append(test_loss)

      x = x.cpu().detach().data
      reconstructed = reconstructed.cpu().detach().data

      clear_output(wait=True)
```

```
      fig, axes = plt.subplots(2, 10, figsize=(16,8))
      fig.suptitle(f'{epoch+1} / {epochs} - train_loss: {train_loss:.4f}, test_loss: {te
st_loss:.4f}')

      for ax in axes.flat:
          ax.set_xticks([]); ax.set_yticks([])

      for i, ax in enumerate(axes[0]):
          ax.imshow(x[i].reshape((28,28)), cmap='gray')
          ax.set_title('Real')
      for i, ax in enumerate(axes[1]):
          ax.imshow(reconstructed[i].reshape((28,28)), cmap='gray')
          ax.set_title('VAE')

      plt.show()

  return train_history, test_history
```

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

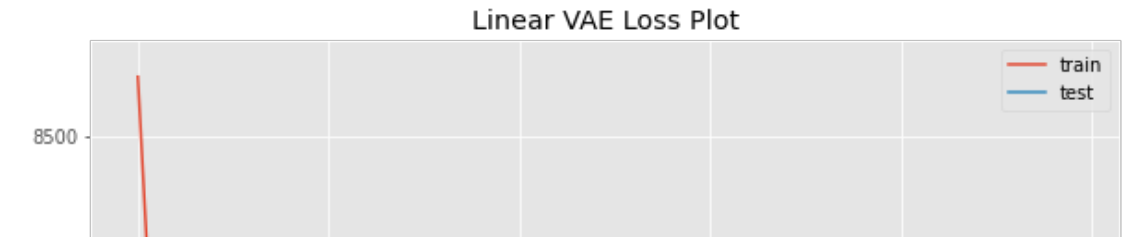
In [44]:

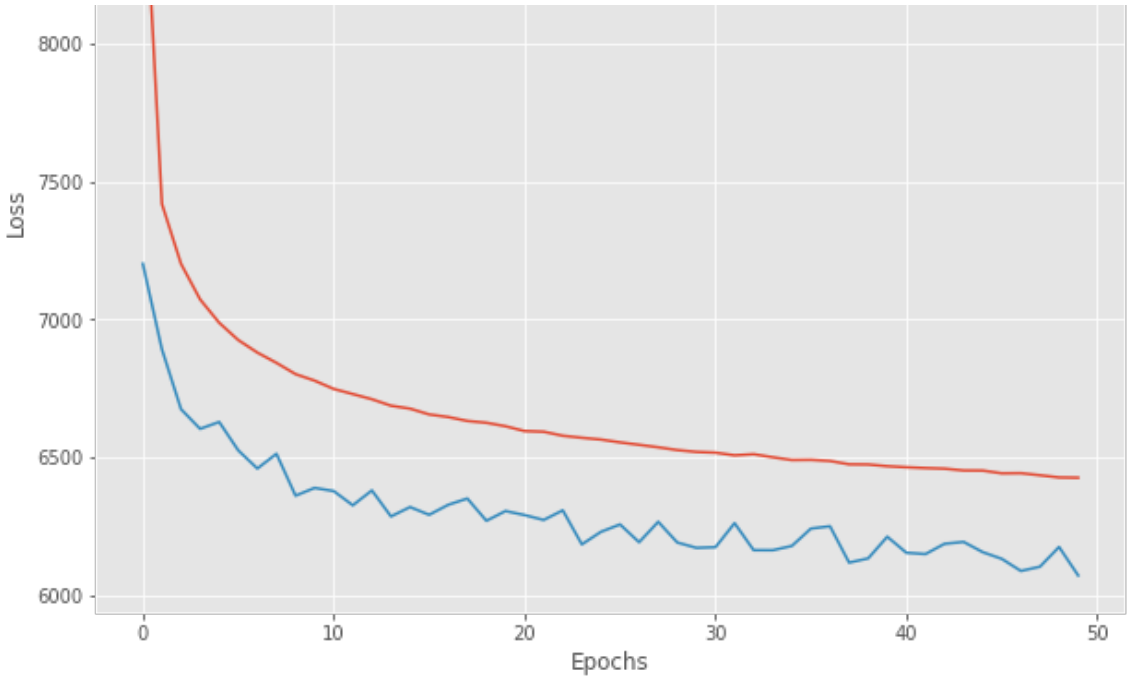
```
train_history, test_history = train_vae(model, optimizer, loss_fn, epochs, train_dataloa
der, test_dataloader)
```



In [45]:

```
plt.figure(figsize=(10, 8))

plt.plot(range(epochs), train_history, label='train')
plt.plot(range(epochs), test_history, label='test')

plt.title('Linear VAE Loss Plot')
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.legend()
plt.show()
```

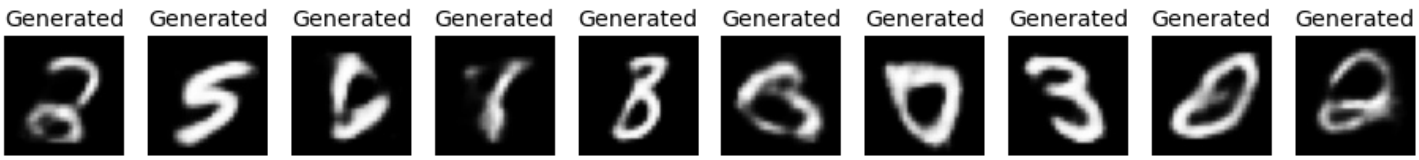


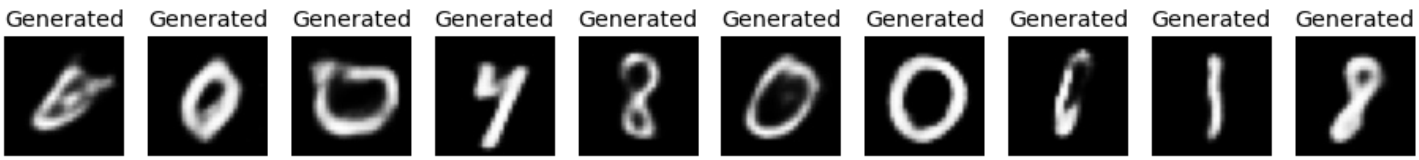


Давайте попробуем проделать для **VAE** то же, что и с обычным автоэнкодером -- подсунуть **decoder**'у из **VAE** случайные векторы из нормального распределения и посмотреть, какие картинки получаются:

In [46]:

```
std_mu, std_sigma = 0, 1
z = std_mu + std_sigma * np.random.randn(24, 16)
z = torch.from_numpy(z.astype('float32')).to(device)
output = model.decode(z).detach().cpu().data

fig, axes = plt.subplots(2, 10, figsize=(16,8))
for i, ax in enumerate(axes.flat):
    ax.imshow(output[i].reshape((28,28)), cmap='gray')
    ax.set_title('Generated')
    ax.axis('off')
```

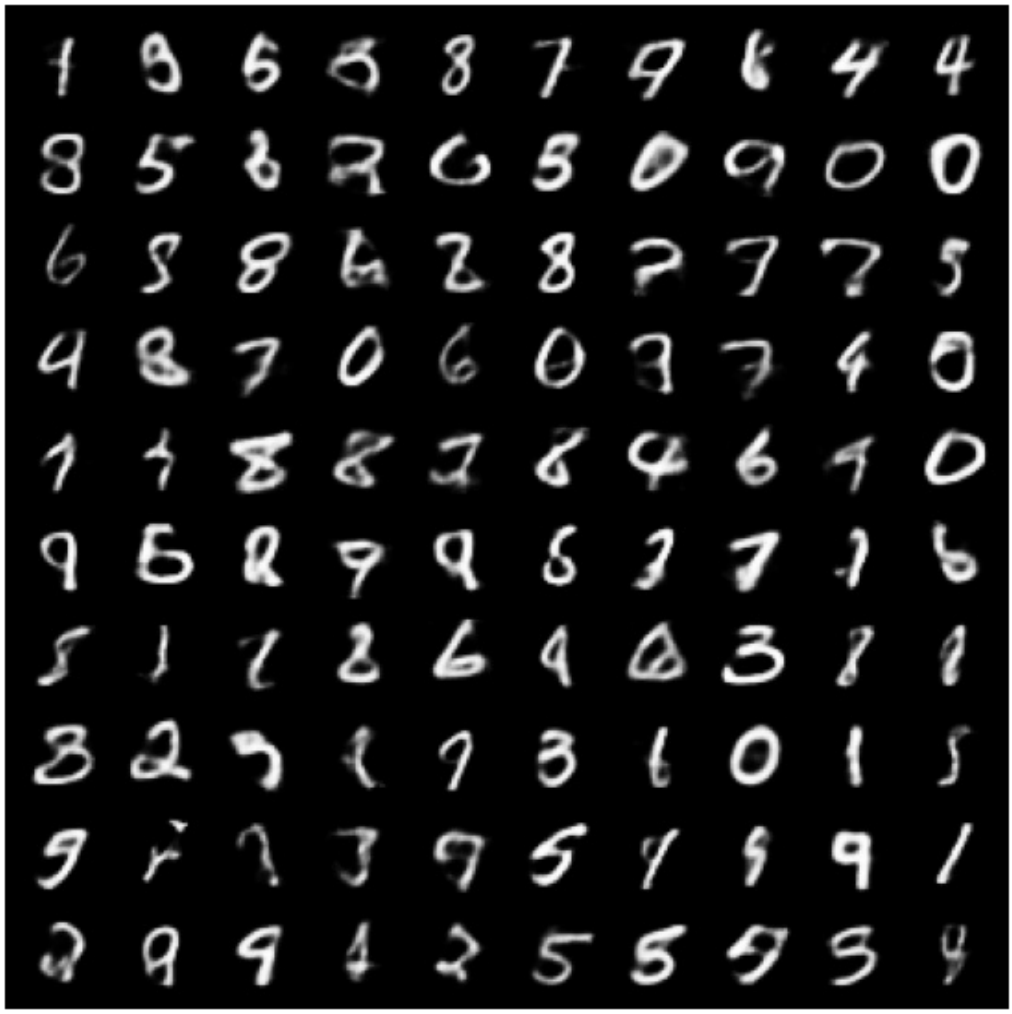




In [47]:

```
import torchvision

model.eval()

def show_image(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))

with torch.no_grad():

    # sample latent vectors from the normal distribution
```

```
    latent = torch.randn(128, 16, device=device)

    # reconstruct images from the latent vectors
    image_reconstructed = model.decode(latent)
    image_reconstructed = image_reconstructed.cpu().detach().data

    fig, ax = plt.subplots(figsize=(10, 10))
    ax.axis('off')
    show_image(torchvision.utils.make_grid(image_reconstructed.data[:100], 10, 5))
    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 просторанстве.

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

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

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

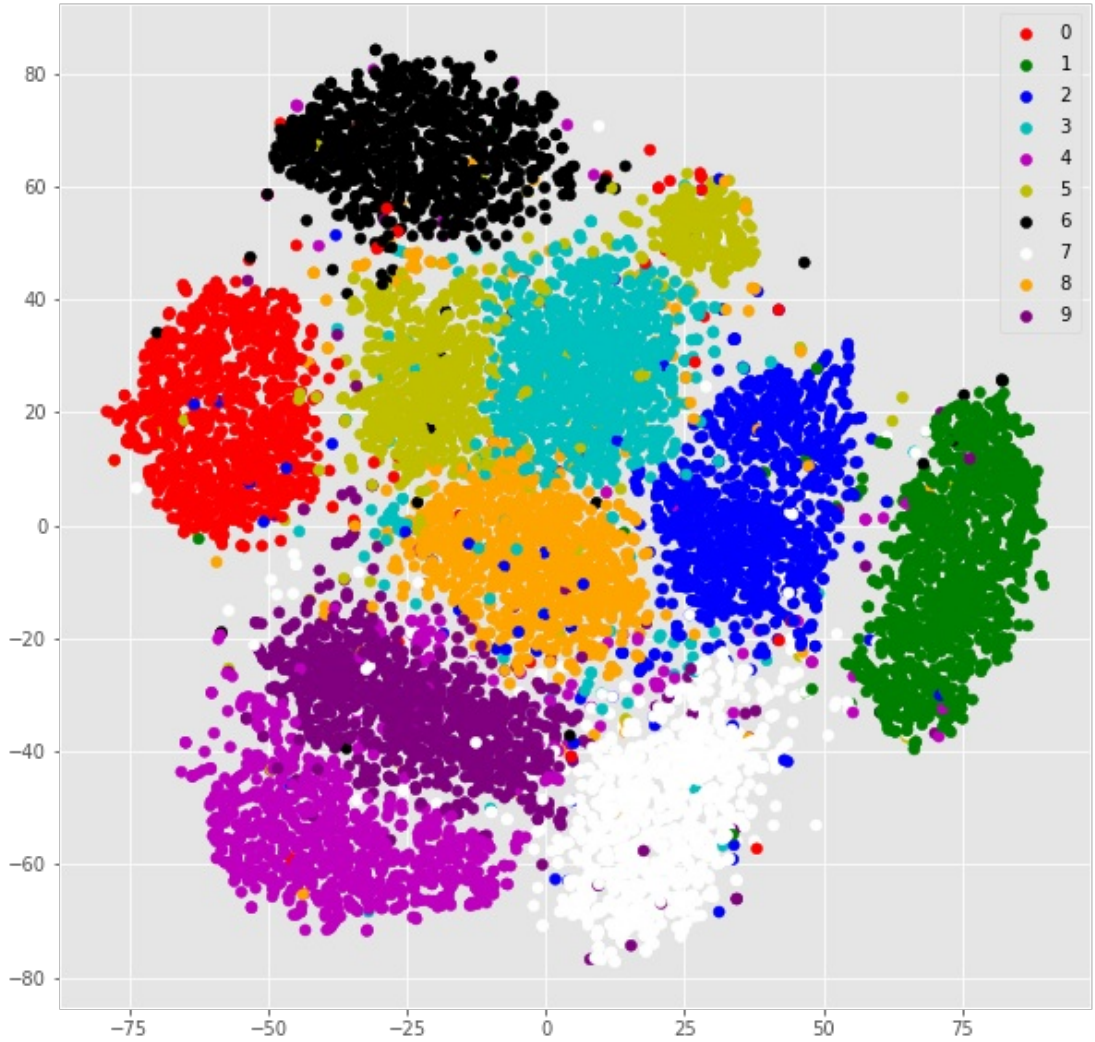
Итак, план:

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

In [48]:

```
%%time
```

```
%%time

from sklearn.manifold import TSNE

z = []

loader = DataLoader(dataset=test_dataset, batch_size=128, shuffle=False)

for x, y in loader:
    x, y = x.to(device), y.to(device)
    z.append(model.encode(x, y)[2])

z = torch.cat(z).detach().cpu().data

tsne = TSNE(n_components=2, random_state=42)
coords = tsne.fit_transform(z)
labels = test_dataset.targets.tolist()

colors = ['r', 'g', 'b', 'c', 'm', 'y', 'k', 'w', 'orange', 'purple']

plt.figure(figsize=(10,10))
scatter = plt.scatter(coords[:,0], coords[:,1], c=labels, cmap=matplotlib.colors.ListedC
olormap(colors))
plt.legend(*scatter.legend_elements())
plt.show()
```




```
CPU times: user 1min 28s, sys: 315 ms, total: 1min 29s
Wall time: 1min 28s
```

Сразу же становятся видны разделимые кластеры чисел от **0** до **9**

2.3. Conditional VAE (6 баллов)

Мы уже научились обучать обычный **AE** на датасете картинок и получать новые картинки, используя генерацию шума и декодер. Давайте теперь допустим, что мы обучили **AE** на датасете **MNIST** и теперь хотим генерировать

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

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

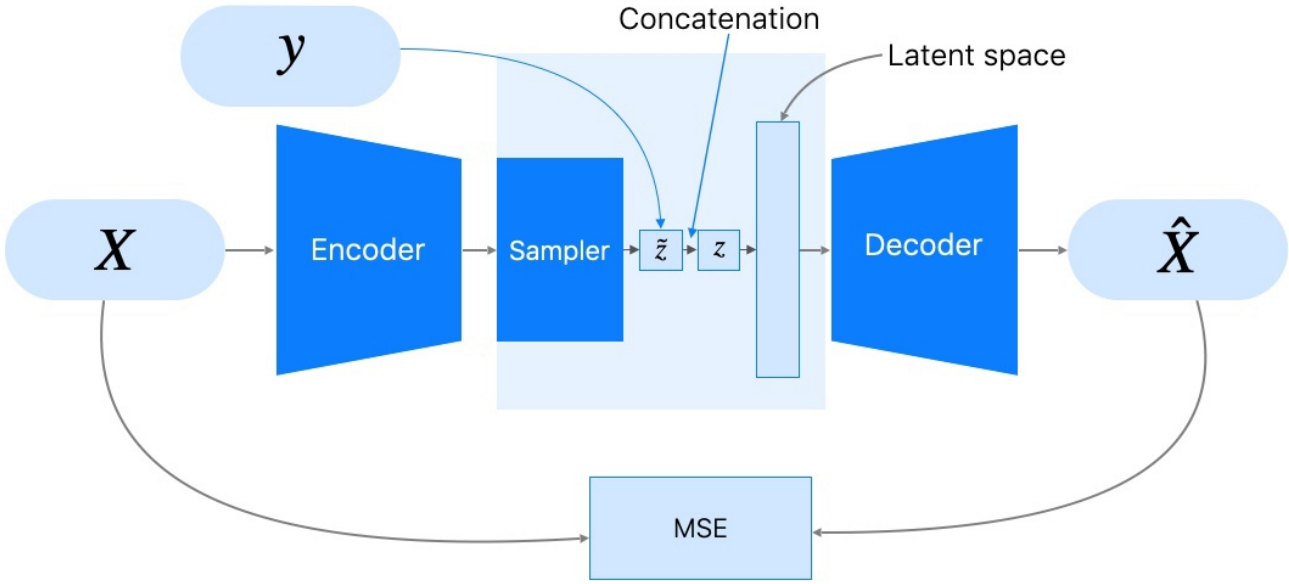
И в этой части задания мы научимся такие обучать.

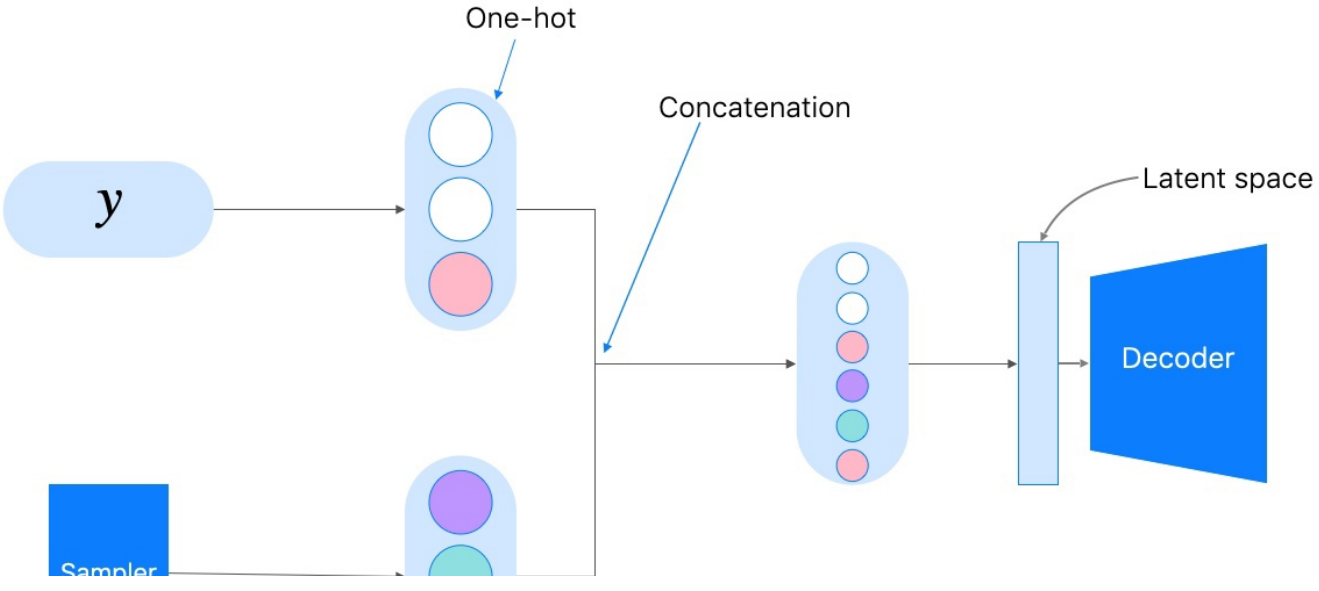
Архитектура

На картинке ниже представлена архитектура простого **Conditional VAE.**

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

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





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

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

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

In [49]:

```
gc.collect()
torch.cuda.empty_cache()
```

In [50]:

```
loss_fn = loss_vae

model = LinearVAE(conditional=True)
model.to(device)

optimizer = optim.AdamW(model.parameters(), lr=1e-3, weight_decay=1e-5)
```

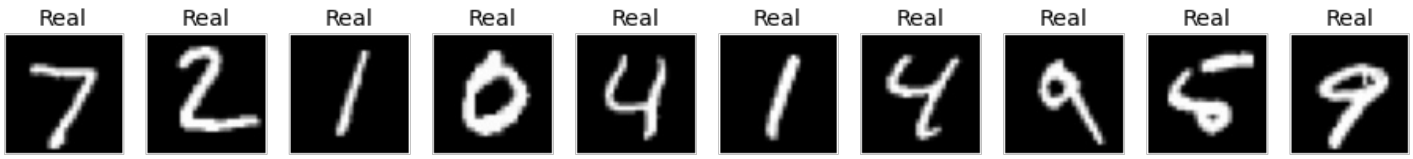
In [51]:

```
epochs=50
```

In [52]:

```
train_history, test_history = train_vae(model, optimizer, loss_fn, epochs, train_dataloa
der, test_dataloader)
```

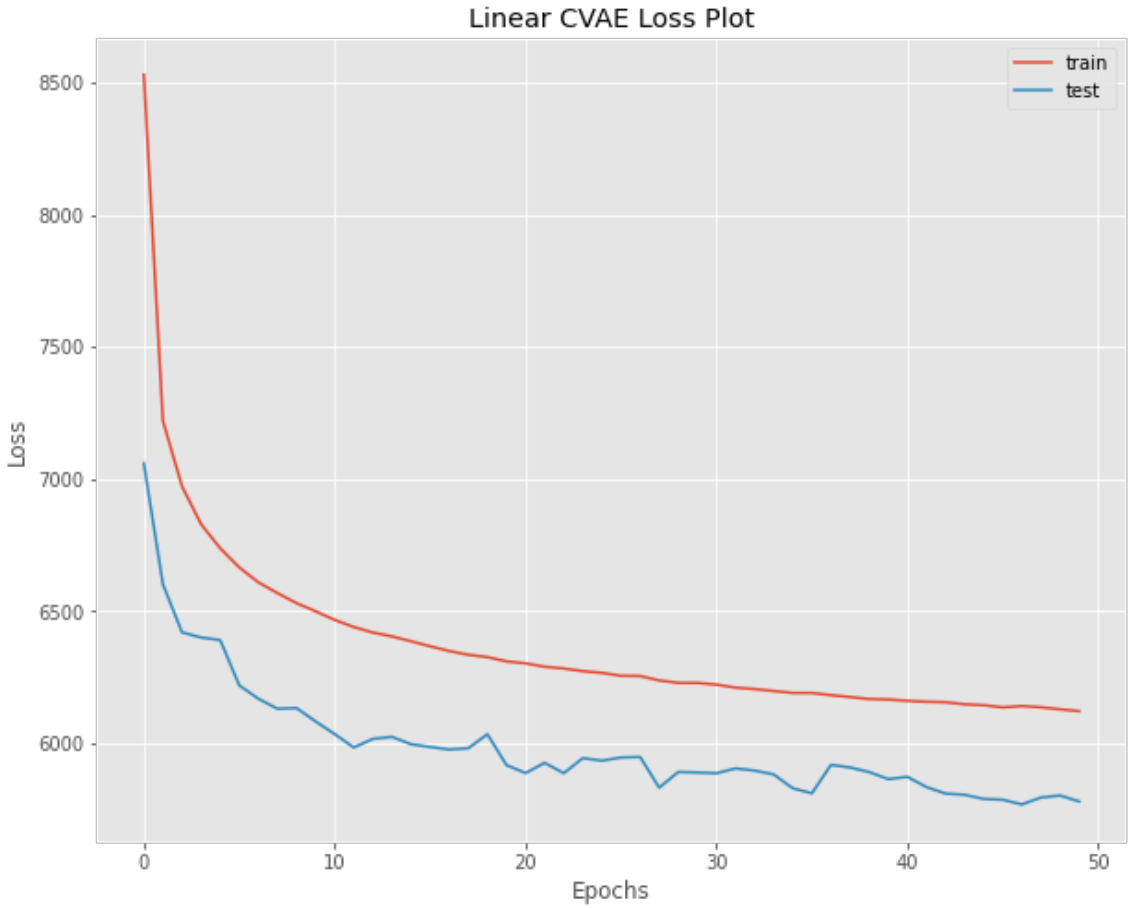
```
50 / 50 - train_loss: 6123.2880, test_loss: 5781.7871
```



In [53]:

```
plt.figure(figsize=(10, 8))

plt.plot(range(epochs), train_history, label='train')
plt.plot(range(epochs), test_history, label='test')
```

```
plt.title('Linear CVAE Loss Plot')
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.legend()
plt.show()
```

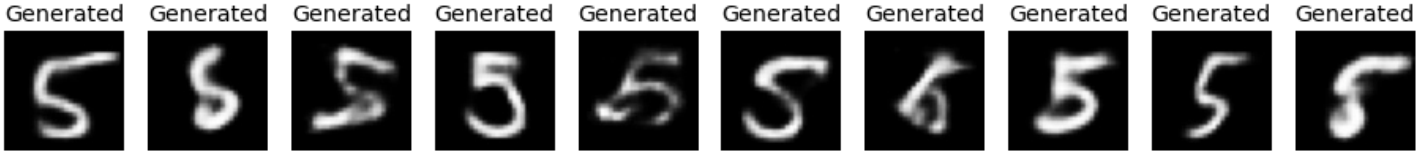


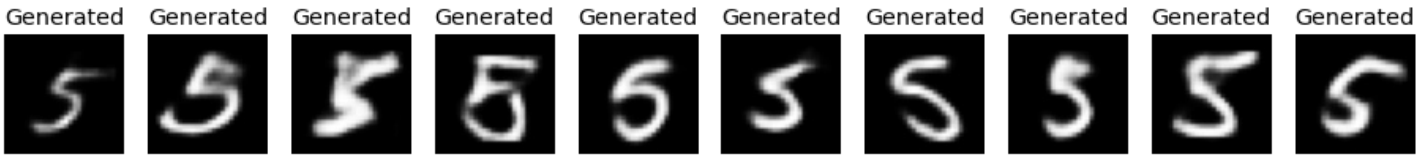
Sampling

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.**

In [54]:

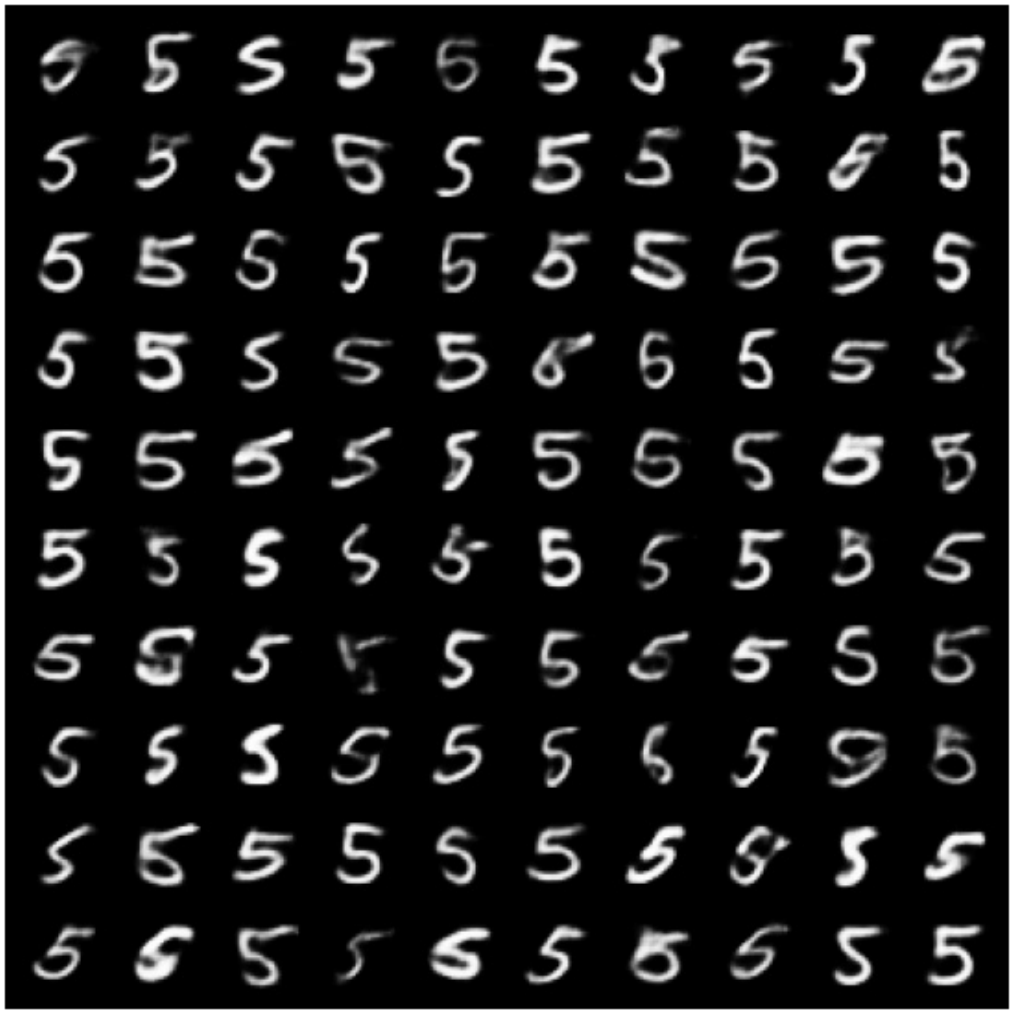
```
std_mu, std_sigma = 0, 1
z = std_mu + std_sigma * np.random.randn(20, 16)
z = torch.from_numpy(z.astype('float32')).to(device)
condition = torch.ones((), dtype=torch.int64).new_full((20,), 5).to(device)

output = model.decode(z, condition).detach().cpu().data


fig, axes = plt.subplots(2, 10, figsize=(16,8))
for i, ax in enumerate(axes.flat):
    ax.imshow(output[i].reshape((28,28)), cmap='gray')
    ax.set_title('Generated')
    ax.axis('off')
```





In [55]:

```
import torchvision

model.eval()

def show_image(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))

with torch.no_grad():

    # sample latent vectors from the normal distribution
    latent = torch.randn(128, 16, device=device)

    # reconstruct images from the latent vectors
    condition = torch.ones((), dtype=torch.int64).new_full((128,), 5).to(device)
    image_reconstructed = model.decode(latent, condition)
    image_reconstructed = image_reconstructed.cpu().detach().data

    fig, ax = plt.subplots(figsize=(10, 10))
    ax.axis('off')
    show_image(torchvision.utils.make_grid(image_reconstructed.data[:100], 10, 5))
    plt.show()
```



Splendid! Вы великолепны!

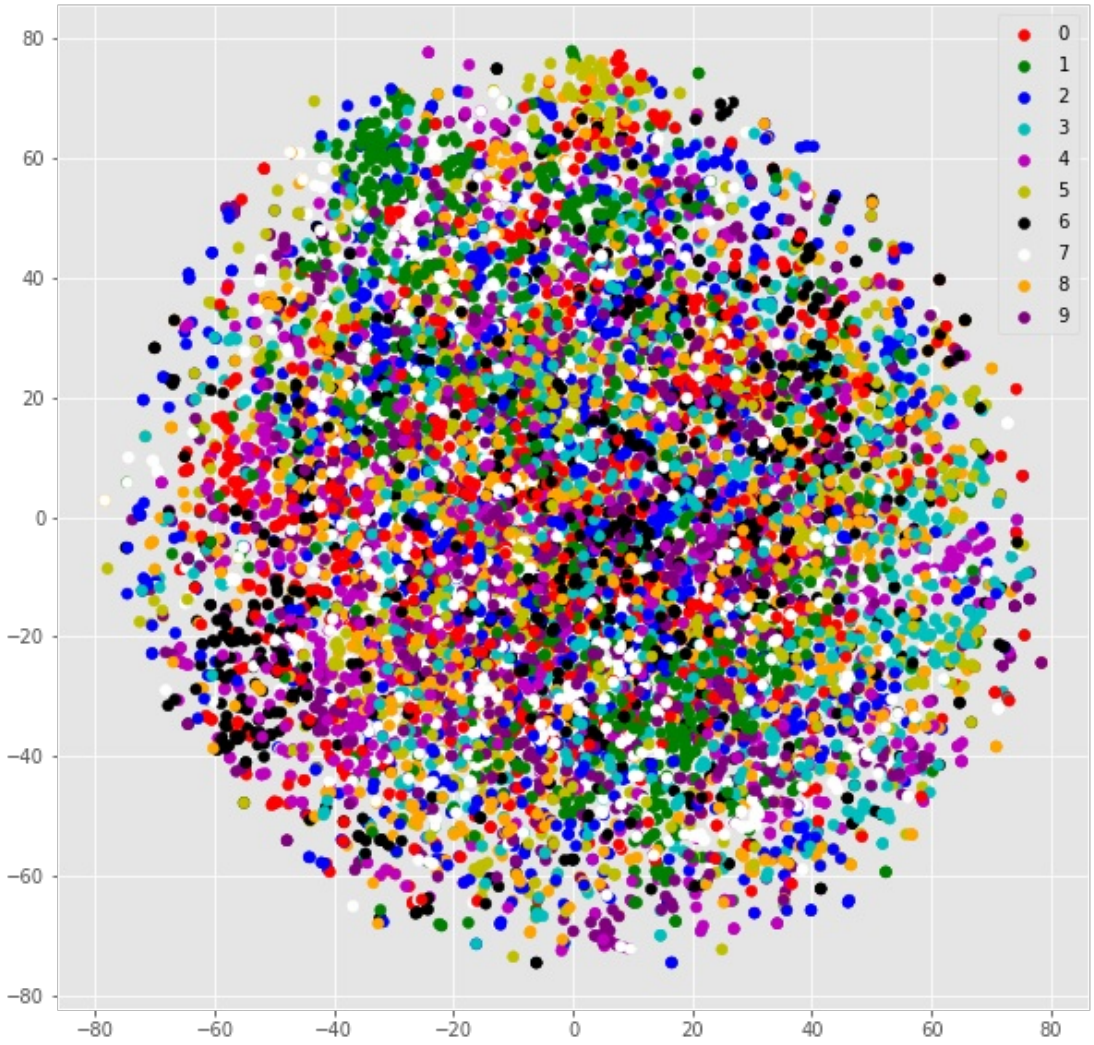
Latent Representations

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.

In [56]:

```
%%time

from sklearn.manifold import TSNE

z = []

loader = DataLoader(dataset=test_dataset, batch_size=128, shuffle=False)

for x, y in loader:
    x, y = x.to(device), y.to(device)
    z.append(model.encode(x, y)[2])

z = torch.cat(z).detach().cpu().data

tsne = TSNE(n_components=2, random_state=42)
coords = tsne.fit_transform(z)
labels = test_dataset.targets.tolist()

colors = ['r', 'g', 'b', 'c', 'm', 'y', 'k', 'w', 'orange', 'purple']

plt.figure(figsize=(10,10))
scatter = plt.scatter(coords[:,0], coords[:,1], c=labels, cmap=matplotlib.colors.ListedC
olormap(colors))
plt.legend(*scatter.legend_elements())
plt.show()
```




```
CPU times: user 1min 30s, sys: 261 ms, total: 1min 30s
Wall time: 1min 30s
```