

Московский государственный технический университет им. Н.Э. Баумана  
Факультет «Информатика и системы управления»  
Кафедра «Системы обработки информации и управления»



**Отчет**  
**Лабораторная работа № 5**  
**По курсу «Проектирование интеллектуальных систем»**

Вариант 9

**ИСПОЛНИТЕЛЬ:**

Попов Илья Андреевич  
Группа ИУ5-23М

\_\_\_\_\_

"\_\_" \_\_\_\_\_ 2022 г.

**ПРЕПОДАВАТЕЛЬ:**

Канев А.И.

\_\_\_\_\_

"\_\_" \_\_\_\_\_ 2022 г.

Москва 2022

---

## **Задание**

Для набора данных с помощью автоэнкодера получить эмбединг изображений и его визуализировать.

Загрузить собственную аудиозапись и использовать автоэнкодер для удаления шума из аудиозаписи.

## **Выполнение**

```
[ ] with open('cifar-100-python/train', 'rb') as f:
    data_train = pickle.load(f, encoding='latin1')
    with open('cifar-100-python/test', 'rb') as f:
        data_test = pickle.load(f, encoding='latin1')

# Здесь указать ваши классы по варианту!!!
CLASSES = [9, 23, 39]

train_X = data_train['data'].reshape(-1, 3, 32, 32)
train_X = np.transpose(train_X, [0, 2, 3, 1]) # NCHW -> NHWC
train_y = np.array(data_train['fine_labels'])
mask = np.isin(train_y, CLASSES)
train_X = train_X[mask].copy()
train_y = train_y[mask].copy()
train_y = np.unique(train_y, return_inverse=1)[1]
del data_train

test_X = data_test['data'].reshape(-1, 3, 32, 32)
test_X = np.transpose(test_X, [0, 2, 3, 1])
test_y = np.array(data_test['fine_labels'])
mask = np.isin(test_y, CLASSES)
test_X = test_X[mask].copy()
test_y = test_y[mask].copy()
test_y = np.unique(test_y, return_inverse=1)[1]
del data_test
Image.fromarray(train_X[50]).resize((256,256))
```



Загружаем датасет по варианту, также используем аугментацию

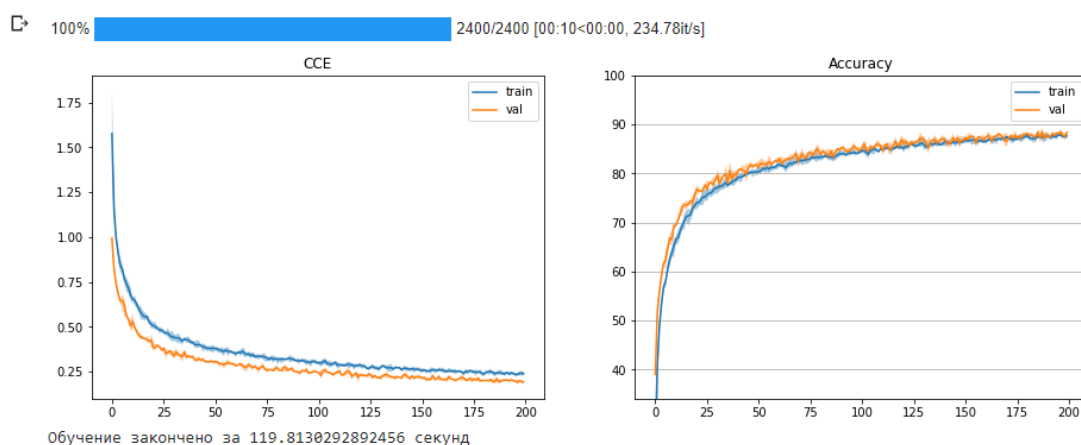
```
# https://blog.jovian.ai/image-classification-of-cifar100-dataset-using-pytorch-8b7145242df1
self.norm = Normalize([0.5074,0.4867,0.4411],[0.2011,0.1987,0.2025])
self.encoder = nn.Sequential(
    nn.Linear(32*32*3, hidden_size),
    nn.ELU(),
    nn.Linear(hidden_size, hidden_size//2),
    nn.ELU(),
    nn.Linear(hidden_size//2, hidden_size//4),
    nn.Tanh()
)
self.decoder = nn.Sequential(
    nn.Linear(hidden_size//4, hidden_size//2),
    nn.ELU(),
    nn.Linear(hidden_size//2, hidden_size),
    nn.ELU(),
    nn.Linear(hidden_size, 32*32*3),
)

def forward(self, input):
    normed = self.norm(input)
    encoded = self.encoder(normed)
    out = self.decoder(encoded)
    return out, encoded, normed

HIDDEN_SIZE = 512
model = Cifar100_AE(hidden_size=HIDDEN_SIZE, classes=len(CLASSES))
model.to(device)
```

```
Cifar100_AE(
  (norm): Normalize()
  (encoder): Sequential(
    (0): Linear(in_features=3072, out_features=512, bias=True)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): ELU(alpha=1.0)
    (4): Linear(in_features=256, out_features=128, bias=True)
    (5): Tanh()
  )
  (decoder): Sequential(
    (0): Linear(in_features=128, out_features=256, bias=True)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=256, out_features=512, bias=True)
    (3): ELU(alpha=1.0)
    (4): Linear(in_features=512, out_features=3072, bias=True)
  )
)
```

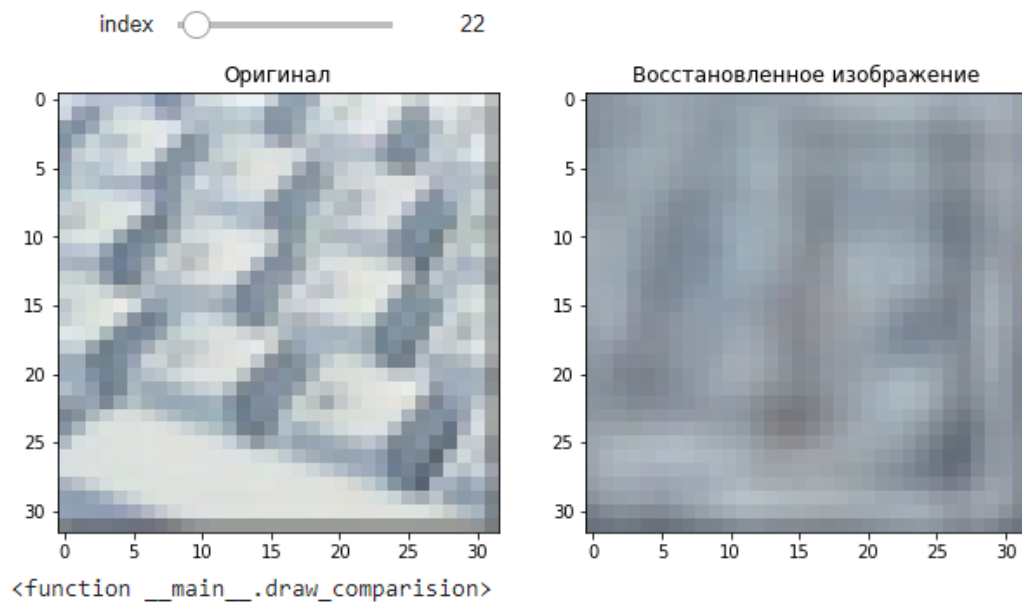
## Определяем автоэнкодер



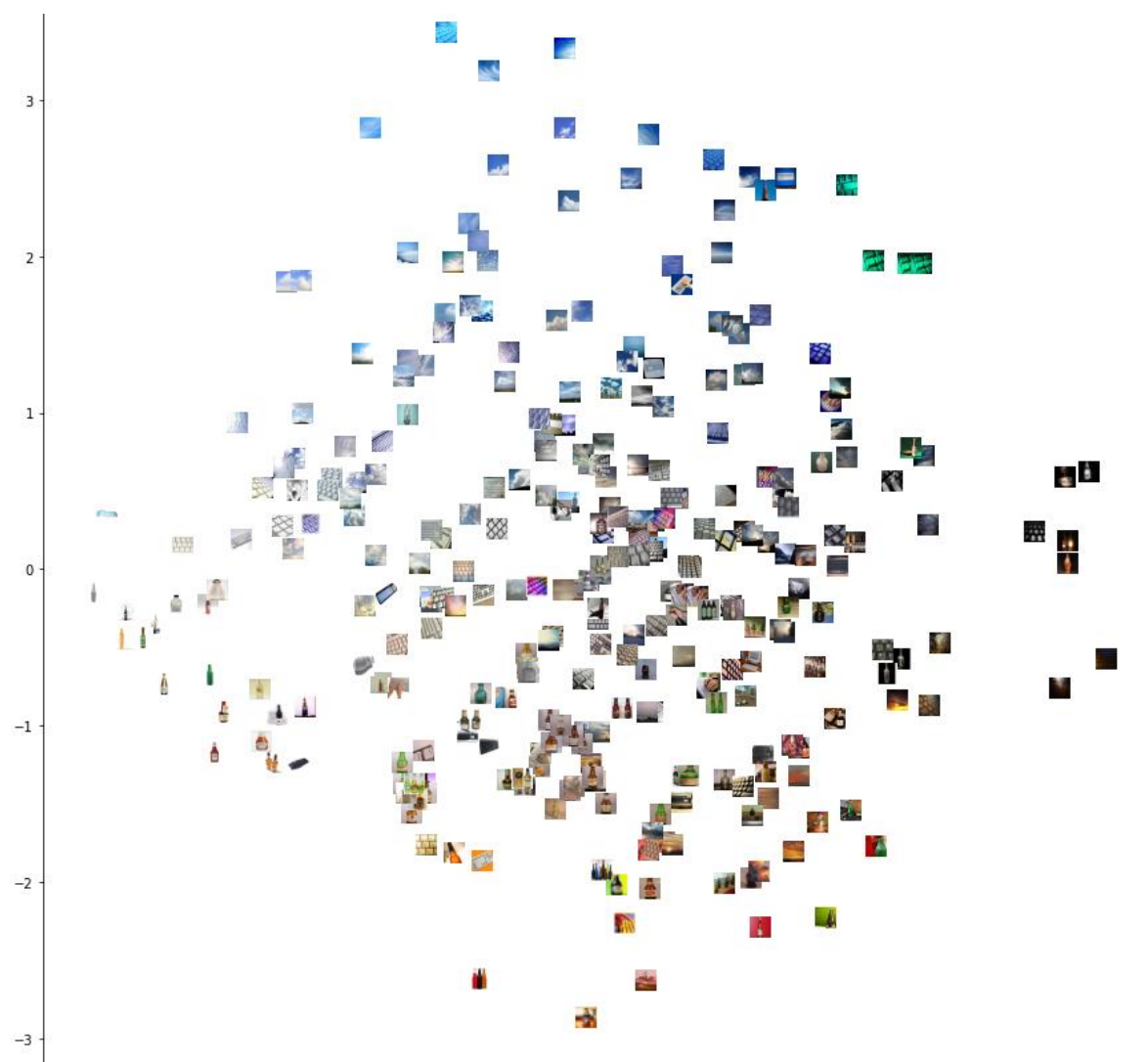
## Обучение модели

## Визуализация восстановленной картинки

```
[ ] def draw_comparision(index=0):  
    fig, ax = plt.subplots(1, 2, figsize=(10, 5))  
    ax[0].imshow(images[index].reshape(32,32,3)/255.)  
    ax[1].imshow(reconstructs[index].reshape(32,32,3))  
    ax[0].set_title('Оригинал')  
    ax[1].set_title('Восстановленное изображение')  
  
    interact(draw_comparision, index=(0, len(images)))
```



Результат работы автоэнкодера



Визуализация эмбединга

## Очистка звука

```
# считывание аудио файла
!ffmpeg -y -i Erik_Satie_G1.mp3 -ac 1 -ar 16000 audio.wav
fs, data = wavfile.read('audio.wav')
#fs, data = wavfile.read('ot_vinta_15_1_24.wav')

data = data / (2**16-1)
data[0] = 1
# добавление шума
#noise = np.random.normal(scale=0.25*data.std(), size=data.shape)
_, noise = wavfile.read('noise.wav')
noise = noise / (2**16-1)
noise = np.tile(noise, 1+data.size//noise.size)[:data.size].copy()
noise = (0.1*data.max()/noise.max())*noise
data_noised = data+noise#np.clip(data + noise, -1, 1)

ffmpeg version 3.4.8-0ubuntu0.2 Copyright (c) 2000-2020 the FFmpeg developers
built with gcc 7 (Ubuntu 7.5.0-3ubuntu1~18.04)
configuration: --prefix=/usr --extra-version=0ubuntu0.2 --toolchain=hardened --libdi
libavutil      55. 78.100 / 55. 78.100
libavcodec     57.107.100 / 57.107.100
libavformat    57. 83.100 / 57. 83.100
libavdevice    57. 10.100 / 57. 10.100
libavfilter    6.107.100 / 6.107.100
libavresample  3.  7.  0 / 3.  7.  0
libswscale     4.  8.100 / 4.  8.100
libswresample  2.  9.100 / 2.  9.100
libpostproc   54.  7.100 / 54.  7.100
[mp3 @ 0x5628de164000] Estimating duration from bitrate, this may be inaccurate
Input #0, mp3, from 'Erik_Satie_G1.mp3':
  Metadata:
    encoder      : Lavf58.20.100
  Duration: 00:03:57.82, start: 0.000000, bitrate: 320 kb/s
```

Загружаем звуковую дорожку, образец шума и объединяем

Звуковая дорожка Erik Satie - Gymnopédie no.1

Образец шума «Дождь»

```
[ ] # https://github.com/digantamisra98/Mish
class Mish(nn.Module):
    def __init__(self):
        super().__init__()
        self.softplus = nn.Softplus()
        self.tanh = nn.Tanh()

    def forward(self, x):
        return x*self.tanh(self.softplus(x))

class DenoisingAE(nn.Module):
    def __init__(self):
        super().__init__()

        self.encoder = nn.Sequential(
            nn.Conv1d(2, 256, kernel_size=3, stride=2, padding=1),
            Mish(),
            nn.Conv1d(256, 512, kernel_size=3, stride=2, padding=1),
            Mish(),
            nn.Conv1d(512, 1024, kernel_size=3, stride=2, padding=1),
            Mish(),
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose1d(1024, 512, kernel_size=3, stride=2, padding=1),
            Mish(),
            nn.ConvTranspose1d(512, 256, kernel_size=3, stride=2, padding=1),
            Mish(),
            nn.ConvTranspose1d(256, 2, kernel_size=3, stride=2, padding=1),
        )

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

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
net = DenoisingAE()
net.to(device)
print(Zxx.shape[0])
net(torch.rand(1, 2, Zxx.shape[0], device=device)).detach().cpu().numpy().shape
```

Определение автоэнкодера



```

▶ scores = []
  scores_val = []
  net.train()
  for epoch in tqdm(range(15)): # loop over the dataset multiple times
      running_loss = 0.0
      for i, batch in tqdm(enumerate(train_dataloader, 0)):
          # get the inputs; data is a list of [inputs, labels]
          inputs, labels = batch
          inputs, labels = inputs.to(device), labels.to(device)


          # zero the parameter gradients
          optimizer.zero_grad()

          # forward + backward + optimize
          outputs = net(inputs)
          loss = criterion(outputs, labels)
          loss.backward()
          optimizer.step()


          # print statistics
          running_loss += loss.item()
      print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / len(train_dataloader):.5f}')
      scores.append(eval_dataset(train_dataloader))
      scores_val.append(eval_dataset(test_dataloader))
      print('R2 score on train:', scores[-1])
      print('R2 score on val:', scores_val[-1])
  print('Finished Training')
  plt.plot(scores, label='training')
  plt.plot(scores_val, label='validation')
  plt.legend()

```

100%  15/15 [02:12<00:00, 8.83s/it]

 105/? [00:06<00:00, 16.56it/s]

[1, 105] loss: 0.02963  
 R2 score on train: -0.2916024524620501  
 R2 score on val: -13.592138623274467

 105/? [00:06<00:00, 16.45it/s]

## Очистка зашумлённой дорожки

Результат очистки лучше проявляется на «спокойных» произведениях