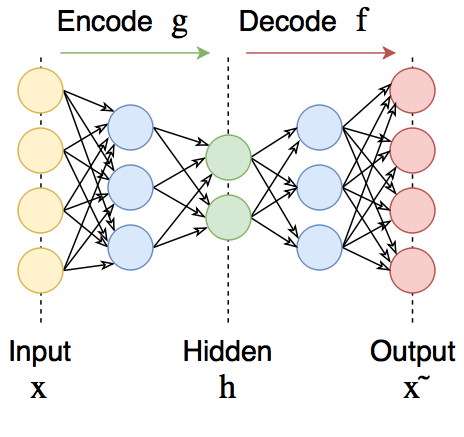
**Применение автоэнкодера для кодирования и декодирования изображений человеческих лиц**

Автоэнкодеры — это нейронные сети прямого распространения, которые восстанавливают входной сигнал на выходе. Внутри у них имеется скрытый слой, который представляет собой код, описывающий модель.

Автоэнкодеры конструируются таким образом, чтобы не иметь возможность точно скопировать вход на выходе. Чтобы достичь такого - их (автоэнкодеры) ограничивают в размерности кода (он меньше, чем размерность сигнала) или штрафуют за активации в коде. Входной сигнал восстанавливается с ошибками из-за потерь при кодировании, но, чтобы их минимизировать, сеть вынуждена учиться отбирать наиболее важные признаки.



Автоэнкодеры состоят из двух частей: энкодера и декодера . Энкодер переводит входной сигнал в его представление (*код*): , в то время как энкодер переводит представление (код) в выходной сигнал, похожий на входной: . Основная задача автоэнкодера - выучить тождественную функцию, т.е.

Алгоритм работы:

1. Подготовка данных (загрузка датасета, обрезка изображений, перекодировка каналов цветов)
2. Нормализация данных (перевод значений каналов цветов в диапазон с 0-255 к 0-1)
3. Создание нейросети и ее обучение
4. Вывод результатов

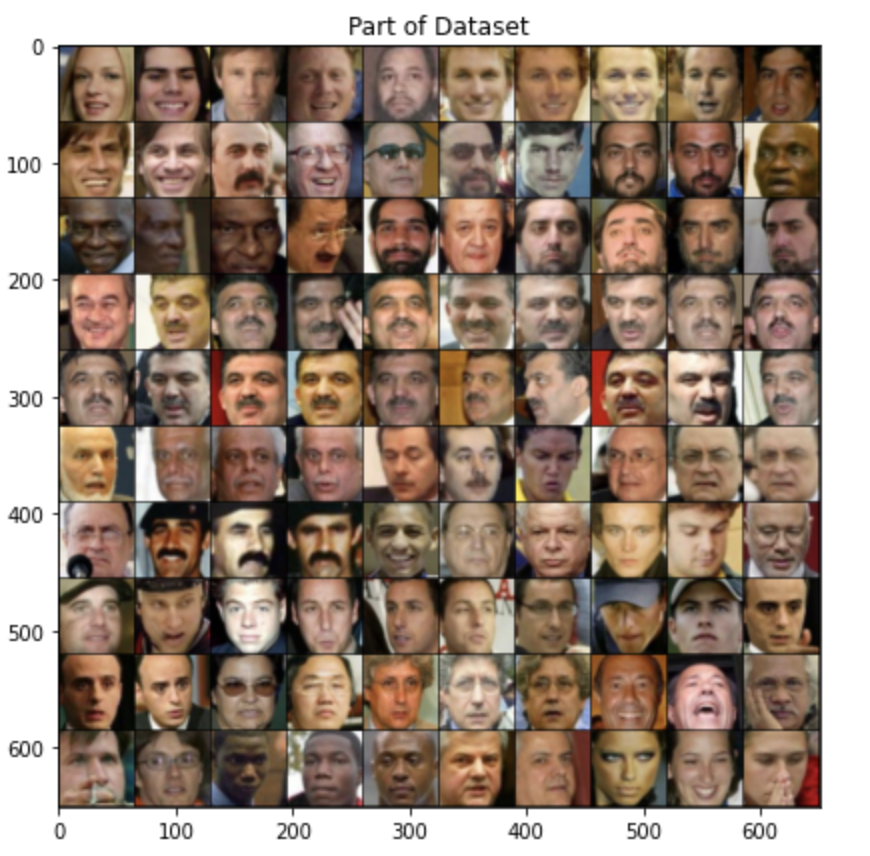
Используемые средства:

* библиотека numpy - для работы с массивами чисел, работа с матрицами и векторами
* библиотека matplotlib - визуализация изображений и графиков
* библиотека cv2 (openCV) - обработка изображений
* библиотека keras - для построения и обучения нейросети. Является высоким уровнем абстракции над tensorflow

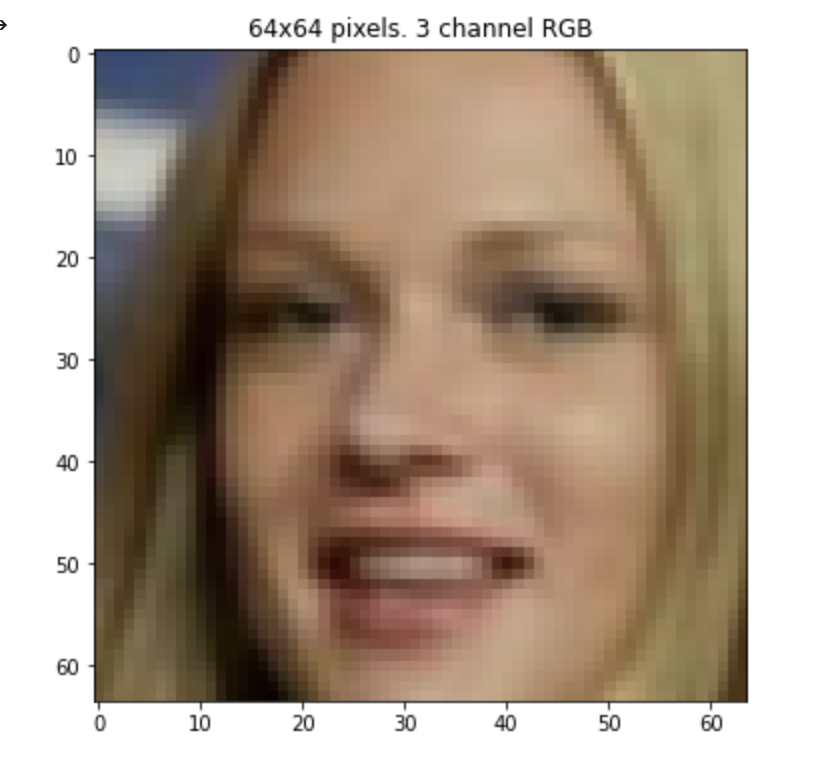
Таблица используемых функций:

| Название | Применение |
| --- | --- |
| cv2.imdecode | Чтение изображения из байтов |
| cv2.cvtColor | Конвертация изображения из одной кодировки цвета в другую |
| cv2.COLOR\_BGR2RGB | Конвертация изображения из кодировки BGR в кодировку RGB |
| cv2.filter2D | Изменяет изображения в зависимости от ядра, переданного в функцию |
| cv2.resize | Изменяет размер изображения |
| np.mean | Считает среднее значение для переданного массива |
| np.std | Считает стандартное отклонение для переданного массива |
| keras.layers.Dense | Полносвязный слой нейросети |
| keras.layers.Flatten | Слой, преобразующий переданный тензор в вектор |
| keras.layers.Reshape | Слой, преобразующий вектор в тензор |
| keras.layers.Input | Инициализирует тензор |
| keras.layers.InputLayer | Слой, являющийся входной точкой в нейросеть |
| keras.models.Sequential | Модель нейросети, являющаяся простой стопкой слоев |
| keras.models.Model | Модель для группировки слоев, которая предоставляет различные интерфейсы, такие как тренировка и др. |

Для датасета был выбран Label Faces in the Wild, состоящий из ~13 тысяч изображений лиц. (<http://vis-www.cs.umass.edu/lfw/>). В последующем изображения обработаны и перекодированы в размер 64x64 пиксела и RGB-модель цвета.



Единичное изображение из датасета:



Модель нейросети представлена следующим образом:

def build\_autoencoder(img\_shape, code\_size):

*"""*

*img\_shape: List[int, int, int] - Ширина, Высота в пикселах и количество каналов цвета*

*code\_size: int - вектор желаемой размерности, который должен создать encoder и в последующем расшифровать decoder*

*"""*

# The encoder

encoder = Sequential()

encoder.add(InputLayer(img\_shape))

encoder.add(Flatten())

encoder.add(Dense(code\_size \* 2))

encoder.add(Dense(code\_size))

# The decoder

decoder = Sequential()

decoder.add(InputLayer((code\_size,)))

decoder.add(Dense(code\_size \* 2))

decoder.add(Dense(np.prod(img\_shape)))

decoder.add(Reshape(img\_shape))

return encoder, decoder

Для обучения выбрана метрика mse - mean squared error. Количество эпох - 30. Желаемая размерность вектора после енкодера - 256.

IMG\_SHAPE = std\_imgs.shape[1:]

encoder, decoder = build\_autoencoder(IMG\_SHAPE, 256)

inp = Input(IMG\_SHAPE)

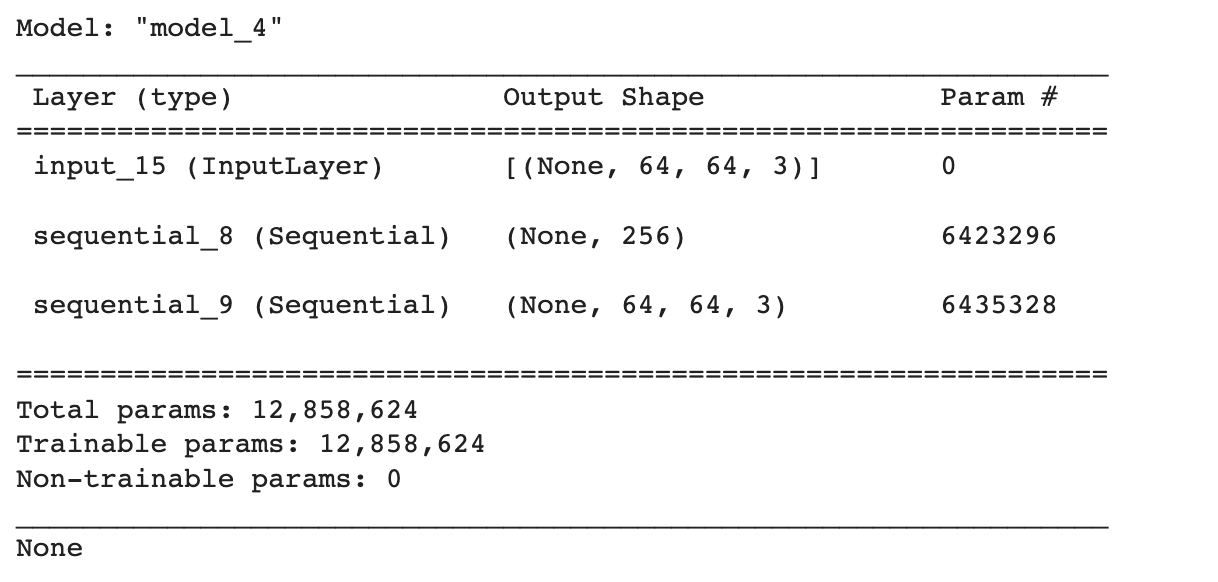
code = encoder(inp)

reconstruction = decoder(code)

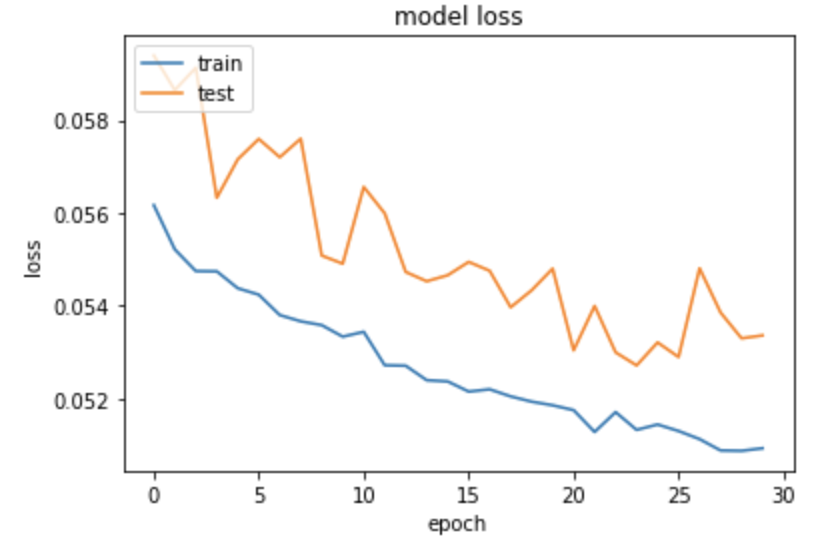
autoencoder = Model(inp,reconstruction)

autoencoder.compile(optimizer='adamax', loss='mse')

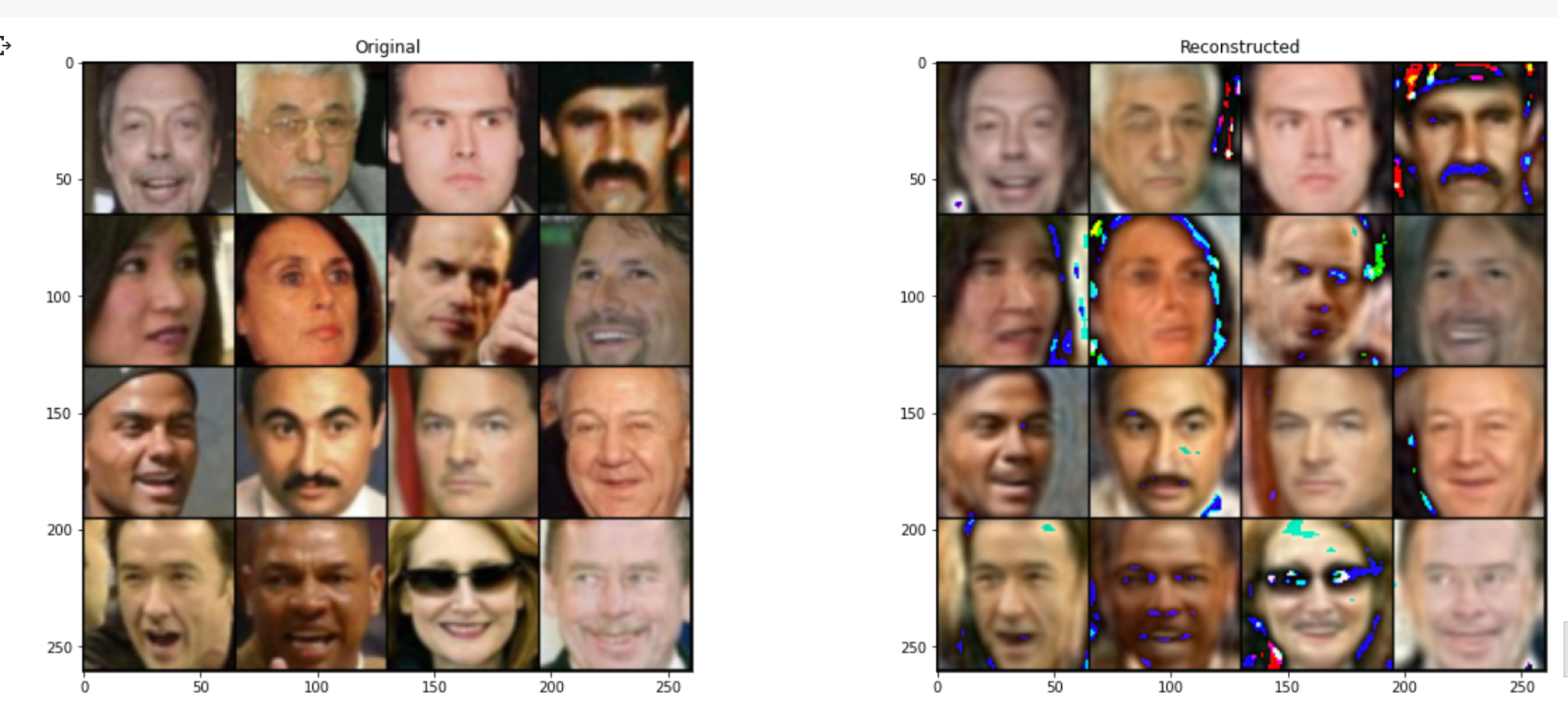
print(autoencoder.summary())



После обучения был построен график потерь:



Проверка компрессии изображений из размерности в 12288 (64 \* 64 \* 3) в 256 и обратно в 12288:



Сама по себе способность автоэнкодеров сжимать данные используется редко, так как обычно они работают хуже, чем вручную написанные алгоритмы для конкретных типов данных. Также для них критически важно, чтобы данные принадлежали той генеральной совокупности, на которой сеть обучалась. Обучив автоэнкодер на цифрах, его нельзя применять для кодирования чего-то другого (например, человеческих лиц).

Листинг программы:

import random

import tarfile

import os

from typing import List, Any

import numpy as np

import matplotlib.pyplot as plt

import cv2

from sklearn.model\_selection import train\_test\_split

import keras

from keras.layers import Dense, Flatten, Reshape, Input, InputLayer

from keras.models import Sequential, Model

os.chdir("/content/data")

os.getcwd()

# http://vis-www.cs.umass.edu/lfw/lfw-deepfunneled.tgz

IMAGES\_NAME = "lfw-deepfunneled.tgz"

def load\_lfw\_dataset(obj\_name, dx=65, dy=65, dimx=45, dimy=45):

imgs = []

with tarfile.open(obj\_name) as f:

for m in f.getmembers():

if m.isfile() and m.name.endswith(".jpg"):

# Чтение

seq = (f.extractfile(m).read())

img = cv2.imdecode(np.asarray(bytearray(seq), dtype=np.uint8), 1)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

# блюр

kernel = (1/9) \* np.array([[1,1,1], [1,1,1], [1,1,1]])

img = cv2.filter2D(img, -1, kernel)

# Обрезка

img = img[dy:-dy, dx:-dx]

img = cv2.resize(img, (dimx, dimy))

imgs.append(img)

return np.array(imgs)

def img\_mean\_pxls(imgs: np.ndarray, axis: int) -> np.ndarray:

*"""*

*Генерирует изображение, где каждый пиксель является усредненным значением*

*всех пикселей переданной выборки в каждой из позиций*

*imgs: выборка изображений*

*axis: по какой размерности усреднять*

*"""*

return np.mean(imgs, axis=axis)

def img\_std\_pxls(imgs: np.ndarray, axis: int) -> np.ndarray:

*"""*

*Генерирует изображение, где каждый пиксель является стандартным отклоенением*

*всех пикселей переданной выборки в каждой из позиций*

*imgs: выборка изображений*

*axis: по какой размерности усреднять*

*"""*

return np.std(imgs, axis=axis)

def preprocess(img, MEAN, STD) -> np.ndarray:

*"""*

*Перевод значения цветов пикселя от формата 0-255 к формату 0-1*

*img: изображение, значения пикселей которых нужно перевести к формату 0-1*

*MEAN: изображение, где каждый пиксель является усредненным значением*

*всех пикселей переданной выборки в каждой из позиций*

*STD: изображение, где каждый пиксель является стандартным отклоенением*

*всех пикселей переданной выборки в каждой из позиций*

*"""*

norm\_img = (img - MEAN) / STD

return norm\_img

def deprocess(norm\_img, MEAN, STD)-> np.ndarray:

*"""*

*Перевод значения цветов пикселя от формата 0-1 к формату 0-255*

*norm\_img: изображение, значения пикселей которых нужно перевести к формату 0-255*

*MEAN: изображение, где каждый пиксель является усредненным значением*

*всех пикселей переданной выборки в каждой из позиций*

*STD: изображение, где каждый пиксель является стандартным отклоенением*

*всех пикселей переданной выборки в каждой из позиций*

*"""*

img = norm\_img \* STD + MEAN

return img.astype(np.uint8)

def img\_table(imgs: np.ndarray, path=None):

*"""*

*Создание таблицы из всех изображений с границей в 1 пиксел.*

*RGB коды в формате 0-255*

*imgs - массив изображений*

*path: str - путь для сохранения картинки в формате png*

*"""*

img\_h, img\_w = imgs.shape[1:3]

n\_plots = int(np.ceil(np.sqrt(imgs.shape[0])))

table = np.ones(

(imgs.shape[1] \* n\_plots + n\_plots + 1,

imgs.shape[2] \* n\_plots + n\_plots + 1,

3))

for i in range(n\_plots):

for j in range(n\_plots):

idx = i \* n\_plots + j

if idx < imgs.shape[0]:

img = imgs[idx]

table[1 + i + i \* img\_h:1 + i + (i + 1) \* img\_h,

1 + j + j \* img\_w:1 + j + (j + 1) \* img\_w] = img

table = table.astype(np.uint8)

if path:

plt.imsave(arr=table, fname=path)

return table

imgs = load\_lfw\_dataset(IMAGES\_NAME, dimx=64, dimy=64)

print("size: {}\nwidth: {}\nheight: {}\ncolor channel: {}".format(\*imgs.shape))

fig, ax = plt.subplots(figsize=(6, 6))

ax.imshow(imgs[0])

plt.title("64x64 pixels. 3 channel RGB")

plt.show()

fig, ax = plt.subplots(figsize=(12, 12))

ax.imshow(img\_table(imgs[:100]))

plt.title("Part of Dataset")

plt.show()

IMG\_MEAN\_PXLS = img\_mean\_pxls(imgs, 0)

IMG\_STD\_PXLS = img\_std\_pxls(imgs, 0)

fig, ax = plt.subplots(figsize=(4, 4))

ax.imshow(IMG\_MEAN\_PXLS.astype(np.uint8))

plt.title("Average value of each pixel")

plt.show()

fig, axs = plt.subplots(1, 2, figsize=(8, 4))

axs[0].imshow(IMG\_STD\_PXLS.astype(np.uint8))

axs[0].set\_title("Standard deviation of each pixel")

axs[1].imshow(np.mean(img\_std\_pxls(imgs, 0).astype(np.uint8), axis=2))

axs[1].set\_title("A clearer identification")

plt.show()

std\_imgs = np.array([

preprocess(img, IMG\_MEAN\_PXLS, IMG\_STD\_PXLS)

for img in imgs

])

x\_train, x\_test = train\_test\_split(std\_imgs, test\_size=0.1, random\_state=42)

len(x\_train), len(x\_test)

def build\_autoencoder(img\_shape, code\_size):

*"""*

*img\_shape: List[int, int, int] - Ширина, Высота в пикселах и количество каналов цвета*

*code\_size: int - вектор желаемой размерности, который должен создать encoder и в последующем расшифровать decoder*

*"""*

# The encoder

encoder = Sequential()

encoder.add(InputLayer(img\_shape))

encoder.add(Flatten())

encoder.add(Dense(code\_size \* 2))

encoder.add(Dense(code\_size))

# The decoder

decoder = Sequential()

decoder.add(InputLayer((code\_size,)))

decoder.add(Dense(code\_size \* 2))

decoder.add(Dense(np.prod(img\_shape)))

decoder.add(Reshape(img\_shape))

return encoder, decoder

IMG\_SHAPE = std\_imgs.shape[1:]

encoder, decoder = build\_autoencoder(IMG\_SHAPE, 256)

inp = Input(IMG\_SHAPE)

code = encoder(inp)

reconstruction = decoder(code)

autoencoder = Model(inp,reconstruction)

autoencoder.compile(optimizer='adamax', loss='mse')

print(autoencoder.summary())

history = autoencoder.fit(x=x\_train, y=x\_train, epochs=30,

validation\_data=[x\_test, x\_test])

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

recs = np.array([

deprocess(autoencoder.predict(img[None])[0], IMG\_MEAN\_PXLS, IMG\_STD\_PXLS)

for img in x\_test

])

fig, axs = plt.subplots(1, 2, figsize=(20, 8))

axs[0].imshow(img\_table(

np.array([

deprocess(img, IMG\_MEAN\_PXLS, IMG\_STD\_PXLS)

for img in x\_test

][:16])

))

axs[0].set\_title("Original")

axs[1].imshow(img\_table(recs[:16]))

axs[1].set\_title("Reconstructed")

plt.show()