**Реализация автодекодера**

Подключаемые библиотеки:

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from pylab import *
```

Инициализация весов модели:

```
def get_init_weights(p, N):
    W = np.random.uniform(size=(N, p), low=-0.02, high=0.02)
    return W, W.T
```

Обработка блока:

```
def process_block(A):
    R = A[:, :, 0].copy()
    G = A[:, :, 1].copy()
    B = A[:, :, 2].copy()
    C = 0.299 * R + 0.587 * G + 0.114 * B
    C_upd = 2.0 * C / 255.0 - 1
    if C_upd.max() > 1:
        raise Exception('Warning: max(C) > 1 and equal to %s' % C_upd.max())
    if C_upd.min() < -1:
        raise Exception('Warning: min(C) < -1 and equal to %s' % C_upd.min())
    return C_upd.reshape(1, C.size)
```

Вычисление коэффициента Z:

```
def get_Z(N, L, p):
    return (float(N) * float(L)) / (float(N + L) * p + 2.0)
```

Восстановление изображения:

```
def restore_pic(A, n, m, hor_blocks, ver_blocks, W, W_):
    P = A.copy()
    for i in range(0, hor_blocks):
        for j in range(0, ver_blocks):
            block = get_block(i, j, n, m, A)
            X_ = process_block(block)
            X0 = X_.reshape(1, X_.size)
            Y = np.dot(X0, W)
            X = np.dot(Y, W_)
            X = (255 * (X[0] + 1) / 2)
            X = X.reshape(1, n * m)
            left = (i * n)
            right = (i + 1) * n
            top = (j * m)
            bottom = (j + 1) * m
            P[left:right, top:bottom, 0] = X[0].reshape(n, m)
            P[left:right, top:bottom, 1] = X[0].reshape(n, m)
            P[left:right, top:bottom, 2] = X[0].reshape(n, m)
    return P
```

Конструктор модели:

```
class Model(object):
    def __init__(self, L=64 * 64, p=20, lr=0.01, e=2, blocks=64, fname='image-test/
square.jpg', adapt=False,
                 norma=False, show_step=10, show=True, max_iter=20, ):
        self.im = Image.open(fname)
        self.pixels_matrix = np.asarray(self.im)
        self.w = self.pixels_matrix.shape[0]
        self.h = self.pixels_matrix.shape[1]
        self.L = L
        self.p = p
```

```
        self.e = e
        self.lr = lr
        self.e = e
        self.adapt_step = adapt
        self.norma = norma
        self.show_step = show_step
        self.show = show
        self.max_iter = max_iter
        self.N = self.h * self.w
        self.block_size = self.N / self.L
        self.hor_blocks, self.ver_blocks = blocks, blocks
        self.n, self.m = self.w / self.hor_blocks, self.h / self.ver_blocks
        self.Z = get_Z(self.block_size, self.L, self.p)
        self.init_weights()
        print ('Blocks: L = %s, n = %s, m = %s, hor_blocks = %s, ver_blocks = %s, Z = %s,
p = %s' %
               (L, self.n, self.m, self.hor_blocks, self.ver_blocks, self.Z, p))
        if self.Z > 1:
            raise Exception('Error: Z > 1')
        if 0.1 * p < e:
            raise Exception('Error: e > 0.1 * p')
        if self.show:
            print ('Before first run')
            print (self.W, self.W_)
            self.draw_pic()
```

Инициализация весов:

```
    def init_weights(self):
        self.W, self.W_ = get_init_weights(self.p, self.block_size)
        self.validate()
```

Вычисление ошибки:

```
    def error(self, X):
        Y = np.matmul(X, self.W)
        X_ = np.matmul(Y, self.W_)
        dX = X_ - X
        return (dX * dX).sum()
```

Шаг обучения для отдельного блока:

```
    def adapt(self, X):
        Y = np.dot(X, self.W)
        X_ = np.dot(Y, self.W_)
        dX = X_ - X
        if self.adapt_step:
            s1 = 1.0 / np.dot(Y, Y.T)
            s2 = 1.0 / np.dot(X, X.T)
            W_t = self.W_ - s1 * np.dot(Y.T, dX)
            Wt = self.W - s2 * np.dot(np.dot(X.T, dX), self.W_.T)
        else:
            W_t = self.W_ - self.lr * np.dot(Y.T, dX)
            Wt = self.W - self.lr * np.dot(np.dot(X.T, dX), self.W_.T)
        self.W = Wt
        self.W_ = W_t
        if self.norma:
            self.W_ /= np.linalg.norm(self.W_, axis=0, keepdims=True)
            self.W /= np.linalg.norm(self.W, axis=1, keepdims=True)
```

Шаг обучения для группы блоков:

```
    def learn_step(self):
        for i in range(0, self.hor_blocks):
            for j in range(0, self.ver_blocks):
                block = get_block(i, j, self.n, self.m, self.pixels_matrix)
                X0 = process_block(block)
                X = X0.reshape(1, X0.size)
```

```
            self.adapt(X)
        E = 0
        for i in range(0, self.hor_blocks):
            for j in range(0, self.ver_blocks):
                block = get_block(i, j, self.n, self.m, self.pixels_matrix)
                X0 = process_block(block)
                X = X0.reshape(1, X0.size)
                E += self.error(X)
        return E
```

Проверка весов:

```
    def validate(self):
        if self.W.max() > 1:
            raise Exception('Warning: max(W) > 1')
        if self.W_.max() > 1:
            raise Exception('Warning: max(W_) > 1')
        if self.W.min() < -1:
            raise Exception('Warning: min(W) < -1')
        if self.W_.min() < -1:
            raise Exception('Warning: min(W_) < -1')
```

Обучение на исходном изображении:

```
    def learn(self):
        iteration = 0
        E = self.e + 1
        errs = []
        while E > self.e and iteration < self.max_iter:
            E = self.learn_step()
            iteration += 1
            errs.append(E)
            if self.show:
                print ('Error: %s, iteration %s' % (E, iteration))
        if self.show:
            plt.plot(errs)
            plt.xlabel("Epoch")
            plt.ylabel("Summary error")
            plt.show()
            print ('After learn')
            print (self.W, self.W_)
            self.draw_pic()
        return E, iteration, errs
```

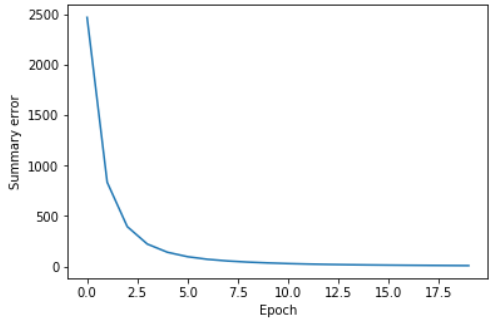
Восстановление изображения:

```
    def draw_pic(self):
        if self.show:
            P = restore_pic(self.pixels_matrix, self.n, self.m, self.hor_blocks,
self.ver_blocks, self.W, self.W_)
            img = Image.fromarray(P)
            plt.imshow(img)
            plt.show()
```

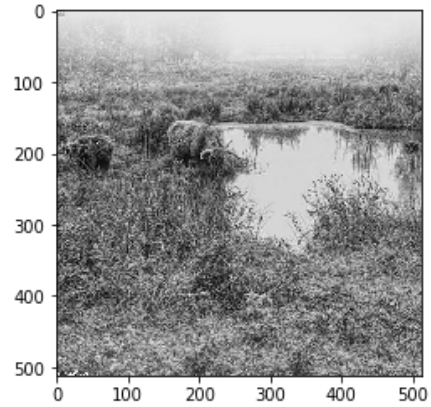
### Базовое тестирование работы автодекодера

### С фиксированным шагом и ненормированными весами:

```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
```



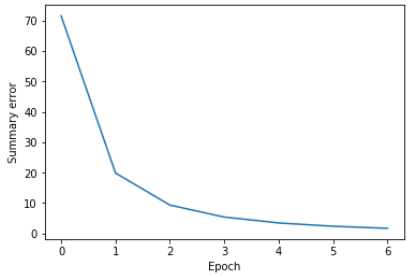

```
After learn
(array([[ 0.04307849,  0.05091241, -0.01663879, ...,  0.15324635,
         0.09621264, -0.11390724],
       [ 0.00903232,  0.09229706,  0.05197822, ..., -0.07269325,
         0.06196727, -0.08015385],
       [ 0.10975354,  0.02892112, -0.04555523, ...,  0.01948537,
         0.01429882,  0.10761648],
       ...,
       [-0.01963782, -0.08490516, -0.09025571, ..., -0.04214733,
        -0.04890749,  0.0080034 ],
       [-0.11390365,  0.08573234, -0.0136215 , ..., -0.07629192,
         0.00668233, -0.07054295],
       [ 0.03595332,  0.14929413,  0.01707001, ..., -0.13616442,
        -0.06130781,  0.0726617 ]]), array([[ 0.05121614,  0.01793416,  0.11702985, ...,
-0.01815067,
        -0.11350792,  0.03636023],
       [ 0.03438607,  0.0762911 ,  0.01355586, ..., -0.09818391,
         0.07447052,  0.13726937],
       [-0.02733699,  0.04093543, -0.05654437, ..., -0.10370793,
        -0.027016  ,  0.0036367 ],
       ...,
       [ 0.13640602, -0.08801025,  0.00220763, ..., -0.06226204,
        -0.09760766, -0.15665317],
       [ 0.10940058,  0.07519268,  0.02731245, ..., -0.03865476,
         0.01745139, -0.05037085],
       [-0.10577795, -0.07177645,  0.11838963, ...,  0.01646455,
        -0.06243554,  0.07865091]]))
```



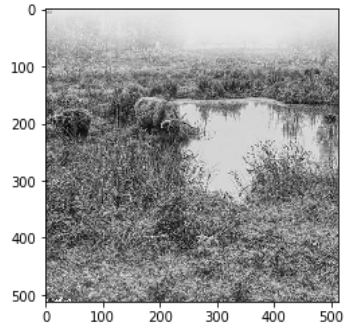

```
e = 9.173066719571773, epochs = 20
```

## С фиксированным шагом и нормированными весами:

```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
```



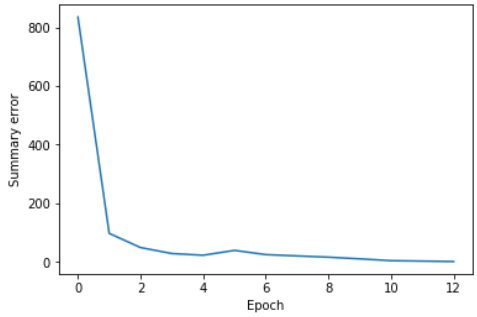

```
After learn
(array([[-0.10581093, -0.01570899,  0.04598234, ...,  0.059937  ,
         0.01598494, -0.12337124],
       [ 0.01615466, -0.11224976,  0.06269169, ...,  0.09621353,
        -0.07612837,  0.02731419],
       [ 0.02504415, -0.03032953, -0.00322035, ..., -0.07517527,
        -0.01950966,  0.03582572],
       ...,
       [-0.04932708,  0.0924133 ,  0.00416593, ...,  0.09829382,
         0.06093459,  0.09234008],
       [-0.03037766,  0.01793772,  0.06422623, ...,  0.00301771,
         0.02662197,  0.02318153],
       [-0.03791645,  0.05754007,  0.00536653, ...,  0.09893076,
        -0.08588293,  0.06935654]]), array([[-0.10444041,  0.01280213,  0.03131866, ...,
-0.04827856,
        -0.03654478, -0.02939523],
       [-0.0128446 , -0.11313503, -0.02731579, ...,  0.0952412 ,
         0.01098676,  0.05910083],
       [ 0.04231174,  0.06067578, -0.00483508, ...,  0.00276086,
         0.06037727,  0.00320056],
       ...,
       [ 0.05886504,  0.09298413, -0.0741795 , ...,  0.09341098,
         0.00636519,  0.10159397],
       [ 0.0133577 , -0.07735937, -0.01783781, ...,  0.05928874,
         0.0276435 , -0.08273406],
       [-0.12354558,  0.02879269,  0.03392351, ...,  0.09329963,
         0.02295327,  0.06688907]]))
```



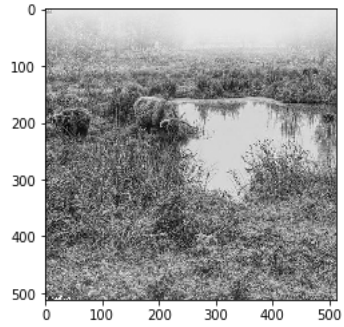
```
e = 1.6981791148242409, epochs = 7
```

## С адаптивным шагом и ненормированными весами:

```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
```



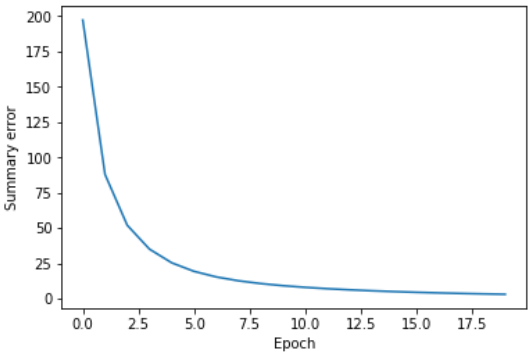

```
After learn
(array([[ 0.01438006,  0.00615835,  0.02764507, ...,  0.00621568,
        -0.10436331, -0.00720211],
       [ 0.08647978,  0.03173822,  0.05316278, ...,  0.00022561,
         0.0362633 ,  0.04031525],
       [-0.02848229,  0.00725584, -0.01602856, ...,  0.0703423 ,
         0.01807097,  0.03707752],
       ...,
       [-0.04675605,  0.02667731, -0.01648492, ..., -0.11267985,
        -0.0043587 , -0.13306094],
       [-0.06007967, -0.06079097, -0.03321535, ...,  0.03151506,
        -0.03916384,  0.12639514],
       [-0.05406916,  0.03876563,  0.02884293, ..., -0.03731861,
         0.02921216,  0.03746313]]), array([[-0.03818689,  0.15287582, -0.0229827 , ...,
-0.2667443 ,
        -0.1286488 , -0.06957225],
       [-0.02134598,  0.02557959, -0.02970361, ...,  0.03440528,
        -0.15567612,  0.15429383],
       [ 0.12678724,  0.1330854 , -0.07294113, ...,  0.03280724,
        -0.17291659,  0.01194953],
       ...,
       [ 0.03764367, -0.00852124,  0.14905308, ..., -0.12844801,
         0.04055022, -0.16398526],
       [-0.15298659,  0.05389467,  0.06239303, ..., -0.0523656 ,
         0.024339  ,  0.06009196],
       [ 0.03526394,  0.16046983,  0.09876652, ..., -0.21409815,
         0.27939565,  0.01456203]]))
```



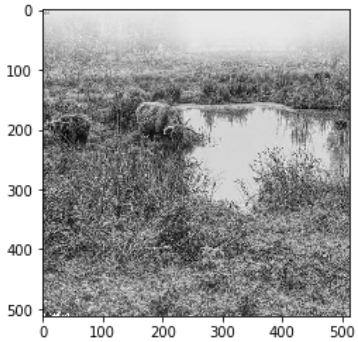
```
e = 1.1371673670659976, epochs = 13
```

## С адаптивным шагом и нормированными весами:

```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
```




```
After learn
(array([[ 0.06242283, -0.06406306, -0.10832845, ...,  0.00061557,
          0.07837213, -0.13045284],
        [ 0.11070107,  0.03867059, -0.00509577, ...,  0.13430782,
         -0.08771632,  0.01435808],
        [-0.05836024,  0.02518447,  0.03674178, ...,  0.10775529,
          0.07773562,  0.0246732 ],
        ...,
        [-0.06192639,  0.07106785,  0.1239928 , ...,  0.04082054,
          0.08887436, -0.0432673 ],
        [-0.13551424,  0.09540985,  0.05090406, ..., -0.10408058,
         -0.00265365, -0.10311767],
        [-0.07331833,  0.0164287 , -0.04388356, ...,  0.02237977,
          -0.04701939,  0.02991014]]), array([[ 0.06395595,  0.10649728, -0.06352323, ...,
-0.04945994,
         -0.14318587, -0.07418278],
        [-0.06010284,  0.04678498,  0.03200037, ...,  0.08330223,
          0.10038789,  0.0233451 ],
        [-0.10908585, -0.0153438 ,  0.01071351, ...,  0.10761478,
          0.06118315, -0.05363564],
        ...,
        [-0.00302776,  0.13420421,  0.10589754, ...,  0.03128148,
         -0.10587586,  0.02932942],
        [ 0.06924328, -0.08085407,  0.09362102, ...,  0.1032623 ,
         -0.01031889, -0.04077116],
        [-0.13147334,  0.01046242,  0.01763535, ..., -0.04136735,
         -0.10590913,  0.02966756]]))
```



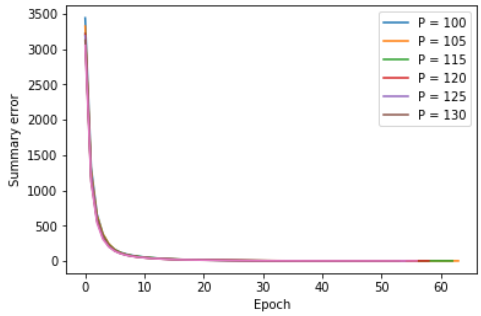

```
e = 3.052206934398979, epochs = 20
```

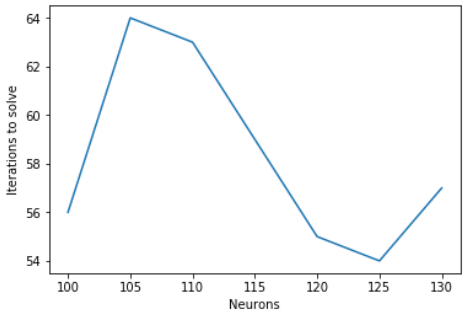
## Тестирование зависимости числа итераций от числа нейронов

## С фиксированным шагом и ненормированными весами:

```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.630150816583, p =
100
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.600143772235, p =
105
```

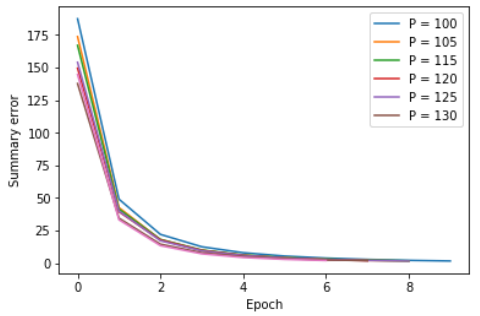
```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.572864629088, p =
110
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.547957575428, p =
115
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.525126101258, p =
120
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.504121137996, p =
125
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.484731935163, p =
130
```

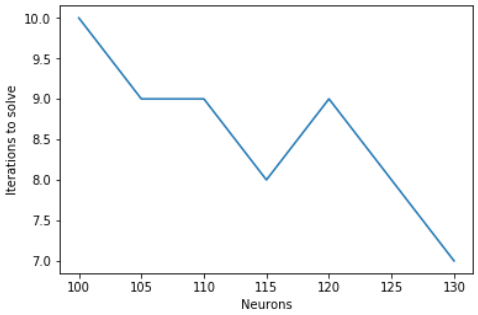



## С фиксированным шагом и нормированными весами:

```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.630150816583, p =
100
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.600143772235, p =
105
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.572864629088, p =
110
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.547957575428, p =
115
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.525126101258, p =
120
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.504121137996, p =
125
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.484731935163, p =
130
```

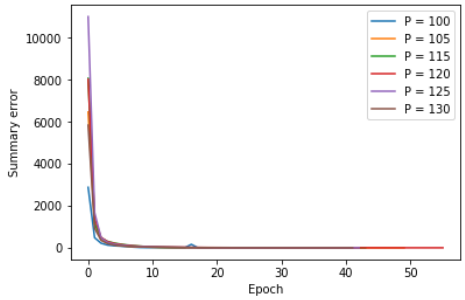


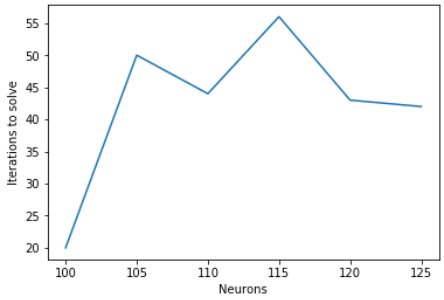




## С адаптивным шагом и ненормированными весами:

```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.630150816583, p =
100
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.600143772235, p =
105
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.572864629088, p =
110
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.547957575428, p =
115
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.525126101258, p =
120
```

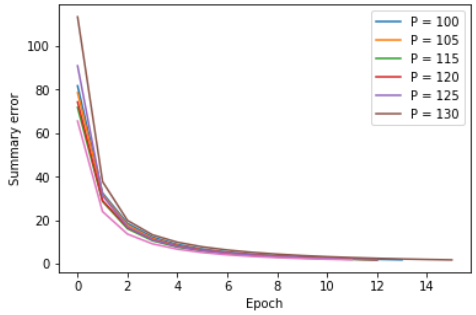
```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.504121137996, p =
125
```

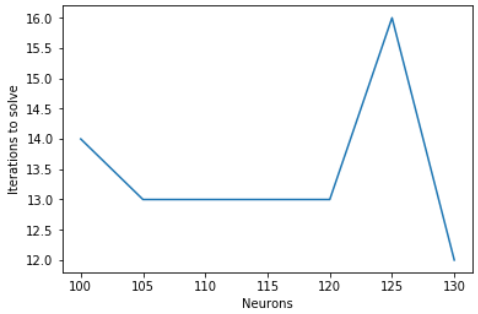



## С адаптивным шагом и нормированными весами:

```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.630150816583, p =
100
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.600143772235, p =
105
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.572864629088, p =
110
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.547957575428, p =
115
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.525126101258, p =
120
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.504121137996, p =
125
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.484731935163, p =
130
```

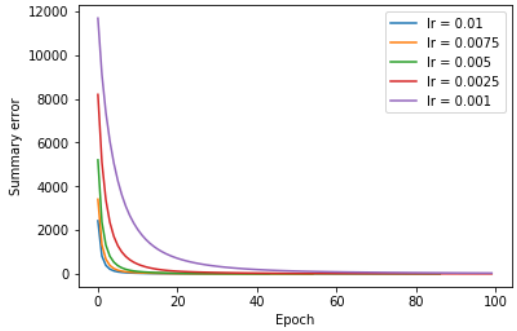


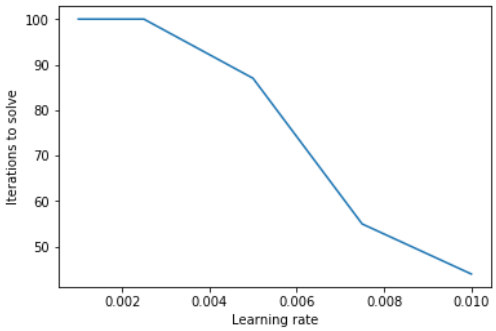




## Тестирование зависимости числа итераций от шага обучения

## С фиксированным шагом и ненормированными весами:

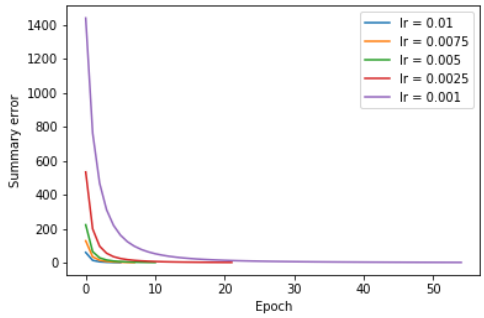
```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
```

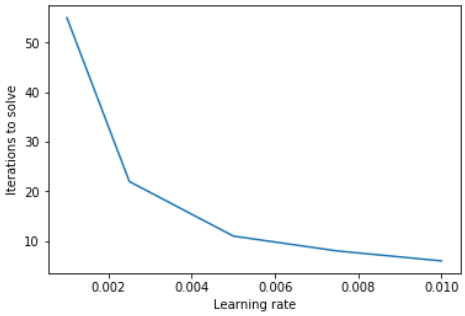



## С фиксированным шагом и нормированными весами:

```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
```

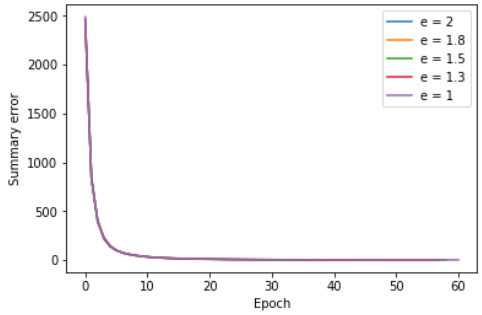


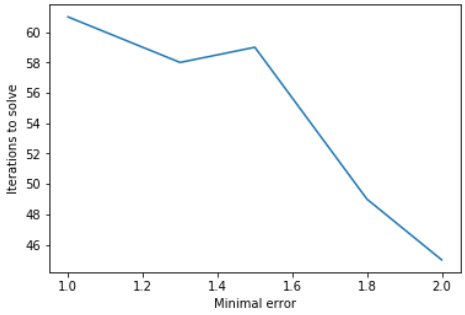




## Тестирование зависимости числа итераций от критерия остановки при достижении минимальной ошибки

## С фиксированным шагом и ненормированными весами:

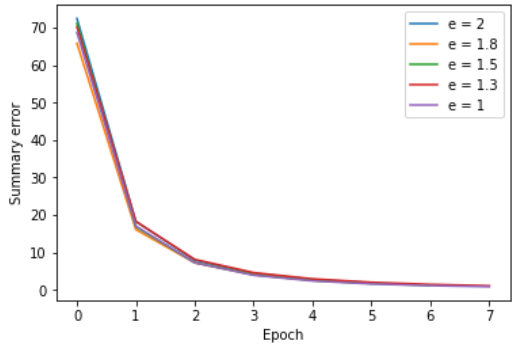
```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
```

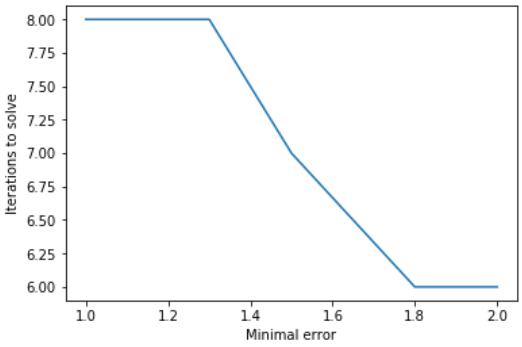



## С фиксированным шагом и нормированными весами:

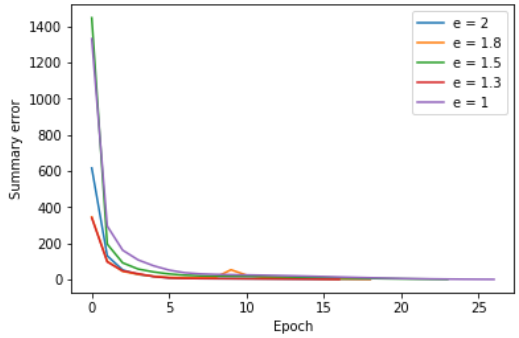
```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
```

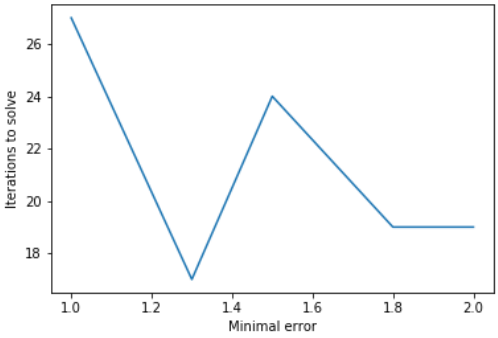



## С адаптивным шагом и ненормированными весами:

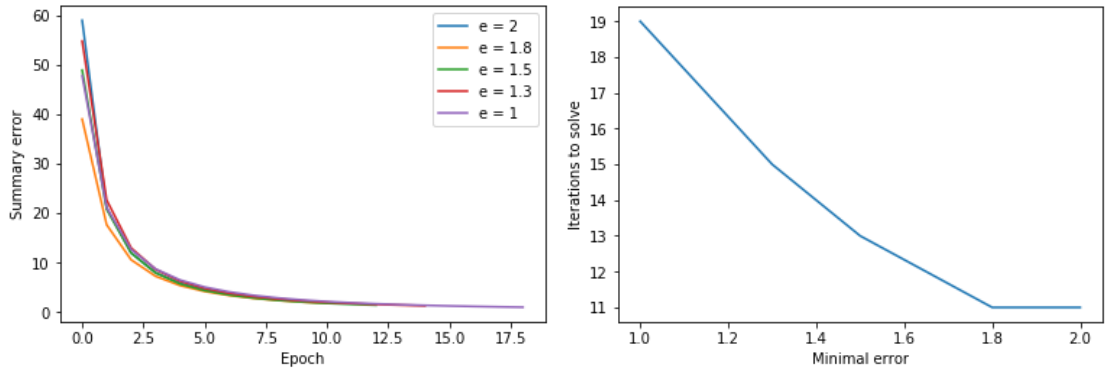
```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
```




## С адаптивным шагом и нормированными весами:

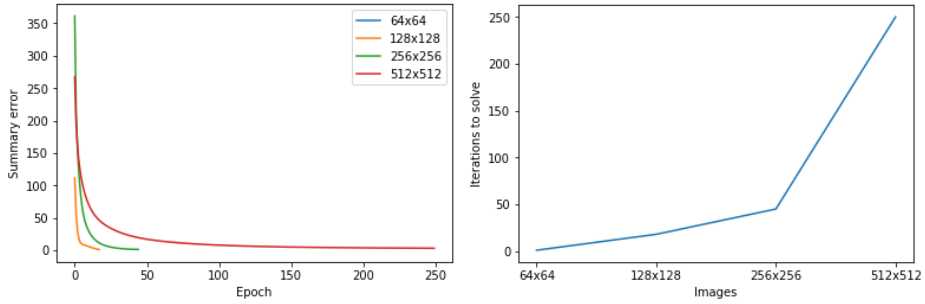
```
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.315076165682, p =
200
```




## Тестирование зависимости числа итераций для разных изображений

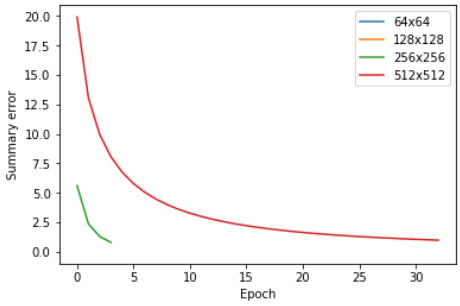
## С ненормированными весами:

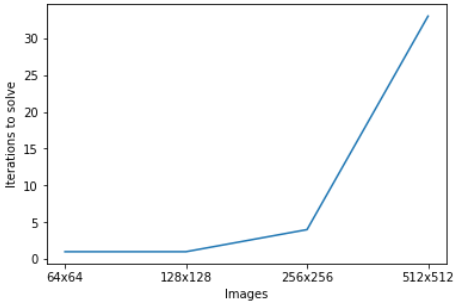
```
Blocks: L = 4096, n = 1, m = 1, hor_blocks = 64, ver_blocks = 64, Z = 0.00399901586719, p
= 250
Blocks: L = 4096, n = 2, m = 2, hor_blocks = 64, ver_blocks = 64, Z = 0.0148003345974, p
= 270
Blocks: L = 4096, n = 4, m = 4, hor_blocks = 64, ver_blocks = 64, Z = 0.0531257245043, p
= 300
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.15753827219, p =
400
```




## С нормированными весами:

```
Blocks: L = 4096, n = 1, m = 1, hor_blocks = 64, ver_blocks = 64, Z = 0.00399901586719, p
= 250
Blocks: L = 4096, n = 2, m = 2, hor_blocks = 64, ver_blocks = 64, Z = 0.0148003345974, p
= 270
Blocks: L = 4096, n = 4, m = 4, hor_blocks = 64, ver_blocks = 64, Z = 0.0531257245043, p
= 300
Blocks: L = 4096, n = 8, m = 8, hor_blocks = 64, ver_blocks = 64, Z = 0.15753827219, p =
400
```






**Выводы:**

Реализована рециркуляционная сеть как модель автодекодера для задачи понижения размерности данных. Получены следующие результаты:

- Алгоритм выполняет меньшее число итераций с увеличением минимальной ошибки;
- Алгоритм быстрее сходится при меньшем шаге адаптивного обучения;
- Применение техник нормирования и адаптивного обучения сокращает число итераций;
- С уменьшением числа входов или выходов уменьшается время работы алгоритма;
- Размерность исходного изображения и размеров блоков влияет прямопропорцианально на время работы алгоритма;
- Применение разнообразных техник работы с матрицами время работы алгоритма может быть сокращено.