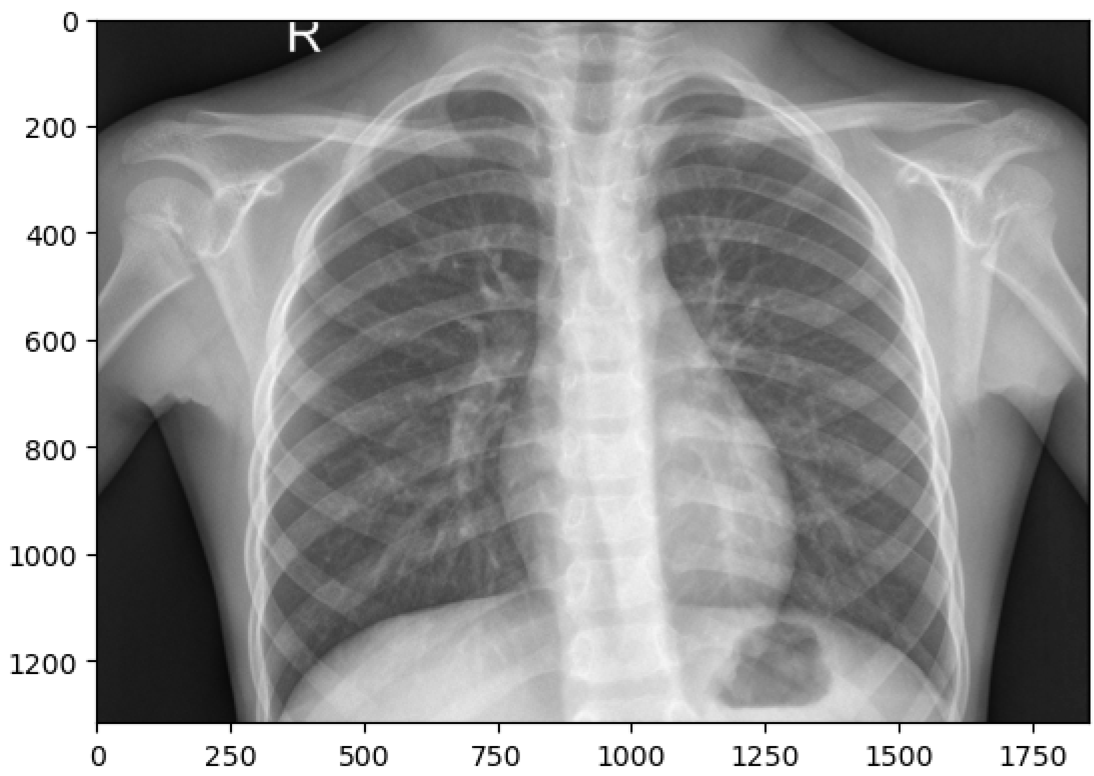
In [1]:
```
from sklearn.datasets import load_sample_image
from sklearn.feature_extraction import image
import numpy as np
from matplotlib import pyplot as plt
import cv2
```

In [14]:
```
# inputting the image from
input_img = "im1_pn_normal.jpeg"

#saving the images that we have into vector variables
img = cv2.imread(input_img,0)
```

In [15]:
```
# the following command will help us understand what the image will look like (vectoriz
img = img/255
# this is going to show us the dimensions of the image  (we can make adjustments based
```

In [16]:
```
plt.imshow(img,cmap='gray')
```

Out[16]: <matplotlib.image.AxesImage at 0x7fd884dd8580>



Looking at the first and last row, we see that there is a black border on the image. To remove this, we can take out the first and last column of the image. Thought the black border is not significant, it is important to remove these things because they add to the external noise that we have. We can do this in the following cell.

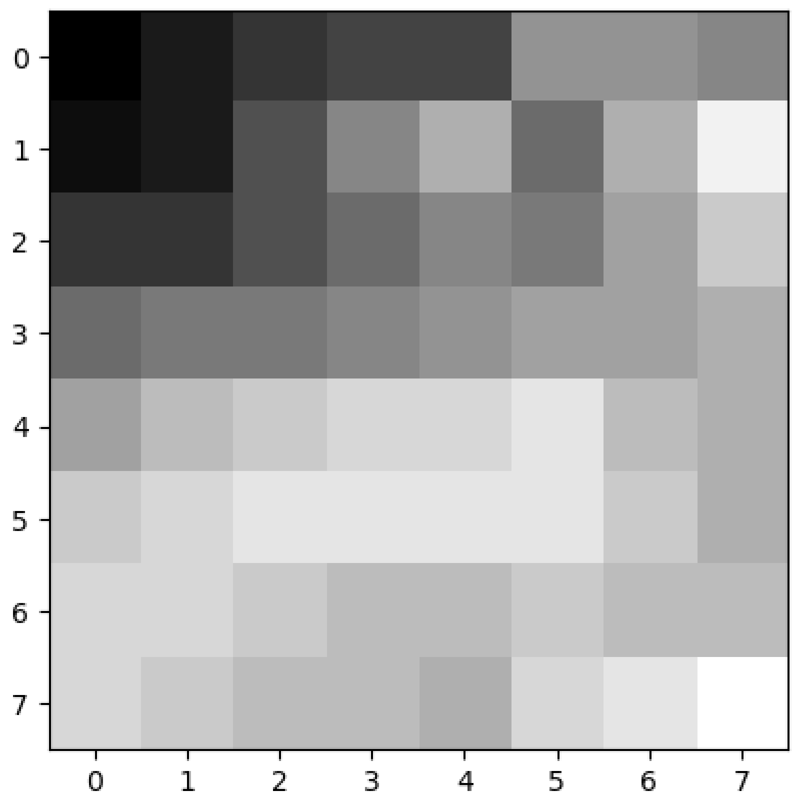
In [18]:
```
k = 8
N = 10000
```

```
patch = image.extract_patches_2d(img, (k, k), max_patches = N, random_state=None) # cha
patch.shape
```

Out[18]:
```
(10000, 8, 8)
```

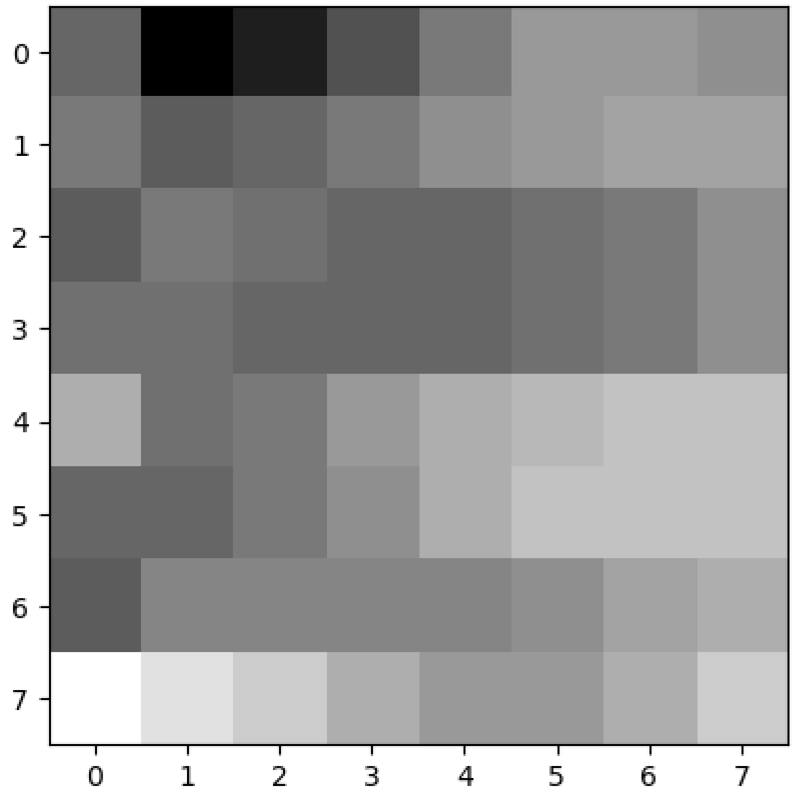
In [19]:
```
import random


plt.imshow(255*patch[random.randint(0,N)],cmap='gray')
```

Out[19]:
```
<matplotlib.image.AxesImage at 0x7fd8400af7f0>
```



In [20]:
```
plt.imshow(patch[0],cmap='gray')
```

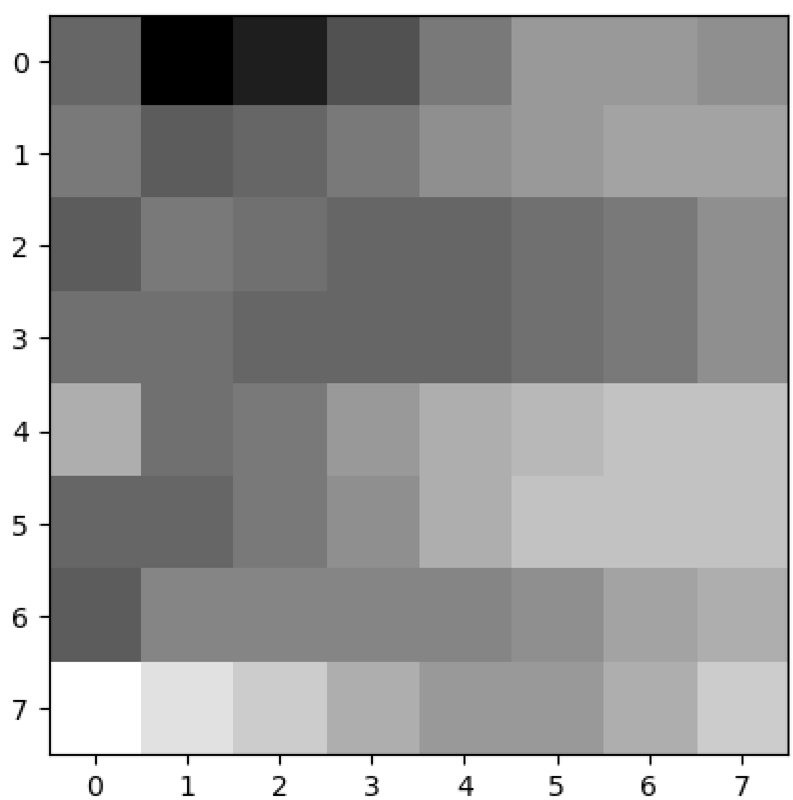
Out[20]:
```
<matplotlib.image.AxesImage at 0x7fd8405d9520>
```



Moving on to the Autoencoder The first thing we would have to do in this place is import our the PyTorch libraries that we are going to use.

In [21]:
```
plt.imshow(patch[0],cmap='gray')
```

Out[21]: <matplotlib.image.AxesImage at 0x7fd840619e50>



In [22]:

```
import torch
from torchvision import datasets
from torchvision import transforms
```

In [23]:

```
patchtensor = torch.from_numpy(patch)
print(patchtensor.data.shape)
type(patchtensor)
```

```
torch.Size([10000, 8, 8])
```

Out[23]:

```
torch.Tensor
```

In [24]:

```
# DataLoader is used to load the dataset
# for training
patchloader = torch.utils.data.DataLoader(dataset = patchtensor, batch_size = 32, shuff
```

The creation of the Autoencoder 1024 = 32*32 ==> 625 (25^2) ==> 400 (20^2) ==> 225 (15^2) ==> 144 (12^2) ==> 121 (11^2) ==> 100

100 ==> 121 ==> 144 ==> 225 ==> 400 ==> 625 ==> 32*32 = 1024

In [31]:

```
# Creating a PyTorch class
# 28*28 ==> 9 ==> 28*28 # change these values
class AE(torch.nn.Module):
    def __init__(self):
        super().__init__()

        # Building an linear encoder with Linear
        # layer followed by Relu activation function
        # 784 ==> 9

        #grow first and then shrink
        self.encoder = torch.nn.Sequential(
            torch.nn.Linear(k * k, 2000),  # change these values, these are not big eno
            torch.nn.ReLU(),
            torch.nn.Linear(2000, 1000),
            torch.nn.ReLU(),
            torch.nn.Linear(1000, 500),
            torch.nn.ReLU(),
            torch.nn.Linear(500, 200),
            torch.nn.ReLU(),
            torch.nn.Linear(200, 100),
        )

        '''
        what can we do with the compressed form of the nn?
        can we take this nn and put it somewhere else so that it can work as transfer o
        '''

        # Building an linear decoder with Linear
        # layer followed by Relu activation function
        # The Sigmoid activation function
        # outputs the value between 0 and 1
        # 9 ==> 784
        self.decoder = torch.nn.Sequential(
```

```
            torch.nn.Linear(100, 200),
            torch.nn.ReLU(),
            torch.nn.Linear(200, 500),
            torch.nn.ReLU(),
            torch.nn.Linear(500, 1000),
            torch.nn.ReLU(),
            torch.nn.Linear(1000, 2000),
            torch.nn.ReLU(),
            torch.nn.Linear(2000, k * k),
            torch.nn.ReLU()
        )

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

In [34]:

```
# Model Initialization
model = AE()

# Validation using MSE Loss function
loss_function = torch.nn.MSELoss()

# Using an Adam Optimizer with lr = 0.1
optimizer = torch.optim.Adam(model.parameters(),lr = 0.0001,
                             weight_decay = 1e-8)
```

In [35]:

```
epochs = 300 #change the epoch value to be larger
outputs = []
losses = []
for epoch in range(epochs):
#     print(epoch)
    for image in patchloader:
        image = image.reshape(-1, k*k)# Reshaping the image to (-1, 784)
        image = image.float()

    # Output of Autoencoder
        reconstructed = model(image)

    # Calculating the loss function
        loss = loss_function(reconstructed, image)

    # The gradients are set to zero,
    # the gradient is computed and stored.
    # .step() performs parameter update
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # Storing the losses in a list for plotting
        losses.append(loss)
        outputs.append((epochs, image, reconstructed))
    print('epoch [{}/{}], loss:{:.8f}'
          .format(epoch + 1, epochs, loss.data.detach().numpy()))
```

```
epoch [1/300], loss:0.06043266
epoch [2/300], loss:0.07363702
epoch [3/300], loss:0.06667658
```

```
epoch [4/300], loss:0.06946521
epoch [5/300], loss:0.05480300
epoch [6/300], loss:0.09153187
epoch [7/300], loss:0.09710239
epoch [8/300], loss:0.07143620
epoch [9/300], loss:0.05849411
epoch [10/300], loss:0.07478226
epoch [11/300], loss:0.04704268
epoch [12/300], loss:0.04389136
epoch [13/300], loss:0.05445882
epoch [14/300], loss:0.07093597
epoch [15/300], loss:0.03857058
epoch [16/300], loss:0.06691921
epoch [17/300], loss:0.03768803
epoch [18/300], loss:0.03415191
epoch [19/300], loss:0.03639372
epoch [20/300], loss:0.02984835
epoch [21/300], loss:0.02684358
epoch [22/300], loss:0.02178157
epoch [23/300], loss:0.02118569
epoch [24/300], loss:0.01869550
epoch [25/300], loss:0.01947327
epoch [26/300], loss:0.01767271
epoch [27/300], loss:0.02130970
epoch [28/300], loss:0.02158368
epoch [29/300], loss:0.02246304
epoch [30/300], loss:0.03109796
epoch [31/300], loss:0.02552317
epoch [32/300], loss:0.02394007
epoch [33/300], loss:0.02460334
epoch [34/300], loss:0.02673310
epoch [35/300], loss:0.02876100
epoch [36/300], loss:0.03509411
epoch [37/300], loss:0.02268166
epoch [38/300], loss:0.03400944
epoch [39/300], loss:0.02420824
epoch [40/300], loss:0.02609454
epoch [41/300], loss:0.02039990
epoch [42/300], loss:0.02424345
epoch [43/300], loss:0.02781108
epoch [44/300], loss:0.02497715
epoch [45/300], loss:0.02407027
epoch [46/300], loss:0.02031045
epoch [47/300], loss:0.02704968
epoch [48/300], loss:0.01523231
epoch [49/300], loss:0.02009228
epoch [50/300], loss:0.01446681
epoch [51/300], loss:0.02751354
epoch [52/300], loss:0.01763477
epoch [53/300], loss:0.01594668
epoch [54/300], loss:0.01695995
epoch [55/300], loss:0.01660194
epoch [56/300], loss:0.01851367
epoch [57/300], loss:0.01544897
epoch [58/300], loss:0.01656405
epoch [59/300], loss:0.01289714
epoch [60/300], loss:0.02023941
epoch [61/300], loss:0.01458902
epoch [62/300], loss:0.01830057
epoch [63/300], loss:0.01224826
epoch [64/300], loss:0.01391597
epoch [65/300], loss:0.01769313
epoch [66/300], loss:0.01525248
epoch [67/300], loss:0.01191692
epoch [68/300], loss:0.01206964
```

```
epoch [69/300], loss:0.01662768
epoch [70/300], loss:0.01779694
epoch [71/300], loss:0.01381768
epoch [72/300], loss:0.01588172
epoch [73/300], loss:0.01320404
epoch [74/300], loss:0.01658601
epoch [75/300], loss:0.01118073
epoch [76/300], loss:0.01802155
epoch [77/300], loss:0.01454802
epoch [78/300], loss:0.01835728
epoch [79/300], loss:0.01540741
epoch [80/300], loss:0.01411799
epoch [81/300], loss:0.01471657
epoch [82/300], loss:0.01228458
epoch [83/300], loss:0.01302375
epoch [84/300], loss:0.01388317
epoch [85/300], loss:0.01431240
epoch [86/300], loss:0.01898106
epoch [87/300], loss:0.01209424
epoch [88/300], loss:0.01142531
epoch [89/300], loss:0.01678863
epoch [90/300], loss:0.01121590
epoch [91/300], loss:0.01192361
epoch [92/300], loss:0.02076020
epoch [93/300], loss:0.01348683
epoch [94/300], loss:0.02123201
epoch [95/300], loss:0.01628959
epoch [96/300], loss:0.01376015
epoch [97/300], loss:0.01536170
epoch [98/300], loss:0.01616757
epoch [99/300], loss:0.01135400
epoch [100/300], loss:0.01509277
epoch [101/300], loss:0.01457911
epoch [102/300], loss:0.01655940
epoch [103/300], loss:0.01440100
epoch [104/300], loss:0.01427676
epoch [105/300], loss:0.01661600
epoch [106/300], loss:0.01329280
epoch [107/300], loss:0.01415523
epoch [108/300], loss:0.01770743
epoch [109/300], loss:0.01869477
epoch [110/300], loss:0.01417702
epoch [111/300], loss:0.01454665
epoch [112/300], loss:0.01574958
epoch [113/300], loss:0.01568397
epoch [114/300], loss:0.01531146
epoch [115/300], loss:0.01351687
epoch [116/300], loss:0.01863121
epoch [117/300], loss:0.01288692
epoch [118/300], loss:0.01383926
epoch [119/300], loss:0.01601641
epoch [120/300], loss:0.01167274
epoch [121/300], loss:0.01671055
epoch [122/300], loss:0.01368116
epoch [123/300], loss:0.01713118
epoch [124/300], loss:0.01390952
epoch [125/300], loss:0.01872683
epoch [126/300], loss:0.01652819
epoch [127/300], loss:0.01118847
epoch [128/300], loss:0.00992620
epoch [129/300], loss:0.00930020
epoch [130/300], loss:0.00893528
epoch [131/300], loss:0.00911087
epoch [132/300], loss:0.01117819
epoch [133/300], loss:0.00993350
```

```
epoch [134/300], loss:0.00581285
epoch [135/300], loss:0.01148440
epoch [136/300], loss:0.01071466
epoch [137/300], loss:0.01095505
epoch [138/300], loss:0.00997097
epoch [139/300], loss:0.00985604
epoch [140/300], loss:0.00797519
epoch [141/300], loss:0.00940326
epoch [142/300], loss:0.01068884
epoch [143/300], loss:0.01026096
epoch [144/300], loss:0.01074444
epoch [145/300], loss:0.00811826
epoch [146/300], loss:0.01119128
epoch [147/300], loss:0.01048363
epoch [148/300], loss:0.00829208
epoch [149/300], loss:0.00933006
epoch [150/300], loss:0.00806571
epoch [151/300], loss:0.00986709
epoch [152/300], loss:0.00885824
epoch [153/300], loss:0.01018447
epoch [154/300], loss:0.00950012
epoch [155/300], loss:0.00877958
epoch [156/300], loss:0.00926914
epoch [157/300], loss:0.00895143
epoch [158/300], loss:0.00986775
epoch [159/300], loss:0.00856738
epoch [160/300], loss:0.01192955
epoch [161/300], loss:0.00924808
epoch [162/300], loss:0.00881414
epoch [163/300], loss:0.00861970
epoch [164/300], loss:0.01029924
epoch [165/300], loss:0.01121714
epoch [166/300], loss:0.01099220
epoch [167/300], loss:0.01037706
epoch [168/300], loss:0.01069369
epoch [169/300], loss:0.00859347
epoch [170/300], loss:0.01258897
epoch [171/300], loss:0.00693710
epoch [172/300], loss:0.00705645
epoch [173/300], loss:0.00783861
epoch [174/300], loss:0.00821028
epoch [175/300], loss:0.00959958
epoch [176/300], loss:0.01219626
epoch [177/300], loss:0.00952902
epoch [178/300], loss:0.01275324
epoch [179/300], loss:0.00814435
epoch [180/300], loss:0.00722145
epoch [181/300], loss:0.00847398
epoch [182/300], loss:0.01023137
epoch [183/300], loss:0.01161337
epoch [184/300], loss:0.01095173
epoch [185/300], loss:0.01431132
epoch [186/300], loss:0.00997945
epoch [187/300], loss:0.00961621
epoch [188/300], loss:0.01076568
epoch [189/300], loss:0.01230104
epoch [190/300], loss:0.00692524
epoch [191/300], loss:0.01139699
epoch [192/300], loss:0.01468909
epoch [193/300], loss:0.01073266
epoch [194/300], loss:0.00832030
epoch [195/300], loss:0.00942221
epoch [196/300], loss:0.01139021
epoch [197/300], loss:0.00909739
epoch [198/300], loss:0.01222110
```

```
epoch [199/300], loss:0.00812580
epoch [200/300], loss:0.00853276
epoch [201/300], loss:0.01291137
epoch [202/300], loss:0.01053017
epoch [203/300], loss:0.00930548
epoch [204/300], loss:0.01038139
epoch [205/300], loss:0.01111472
epoch [206/300], loss:0.00848706
epoch [207/300], loss:0.01374518
epoch [208/300], loss:0.00875814
epoch [209/300], loss:0.00761574
epoch [210/300], loss:0.00957476
epoch [211/300], loss:0.01051218
epoch [212/300], loss:0.00828928
epoch [213/300], loss:0.01073689
epoch [214/300], loss:0.00936611
epoch [215/300], loss:0.00966227
epoch [216/300], loss:0.00887901
epoch [217/300], loss:0.01141950
epoch [218/300], loss:0.00792487
epoch [219/300], loss:0.00807443
epoch [220/300], loss:0.00751121
epoch [221/300], loss:0.00898845
epoch [222/300], loss:0.01348375
epoch [223/300], loss:0.00813408
epoch [224/300], loss:0.00967126
epoch [225/300], loss:0.00930436
epoch [226/300], loss:0.01061417
epoch [227/300], loss:0.00910169
epoch [228/300], loss:0.01090287
epoch [229/300], loss:0.00819756
epoch [230/300], loss:0.01081769
epoch [231/300], loss:0.00830502
epoch [232/300], loss:0.01074182
epoch [233/300], loss:0.00994542
epoch [234/300], loss:0.01097839
epoch [235/300], loss:0.01047715
epoch [236/300], loss:0.00779127
epoch [237/300], loss:0.01259001
epoch [238/300], loss:0.01194082
epoch [239/300], loss:0.00849503
epoch [240/300], loss:0.00467340
epoch [241/300], loss:0.00418220
epoch [242/300], loss:0.00653149
epoch [243/300], loss:0.00394564
epoch [244/300], loss:0.00332109
epoch [245/300], loss:0.00553579
epoch [246/300], loss:0.00488001
epoch [247/300], loss:0.00449373
epoch [248/300], loss:0.00658412
epoch [249/300], loss:0.00556301
epoch [250/300], loss:0.00612192
epoch [251/300], loss:0.00561865
epoch [252/300], loss:0.00657624
epoch [253/300], loss:0.00437967
epoch [254/300], loss:0.00388434
epoch [255/300], loss:0.00476438
epoch [256/300], loss:0.00512015
epoch [257/300], loss:0.00647693
epoch [258/300], loss:0.00424037
epoch [259/300], loss:0.00415684
epoch [260/300], loss:0.00515364
epoch [261/300], loss:0.00600306
epoch [262/300], loss:0.00641487
epoch [263/300], loss:0.00532728
```

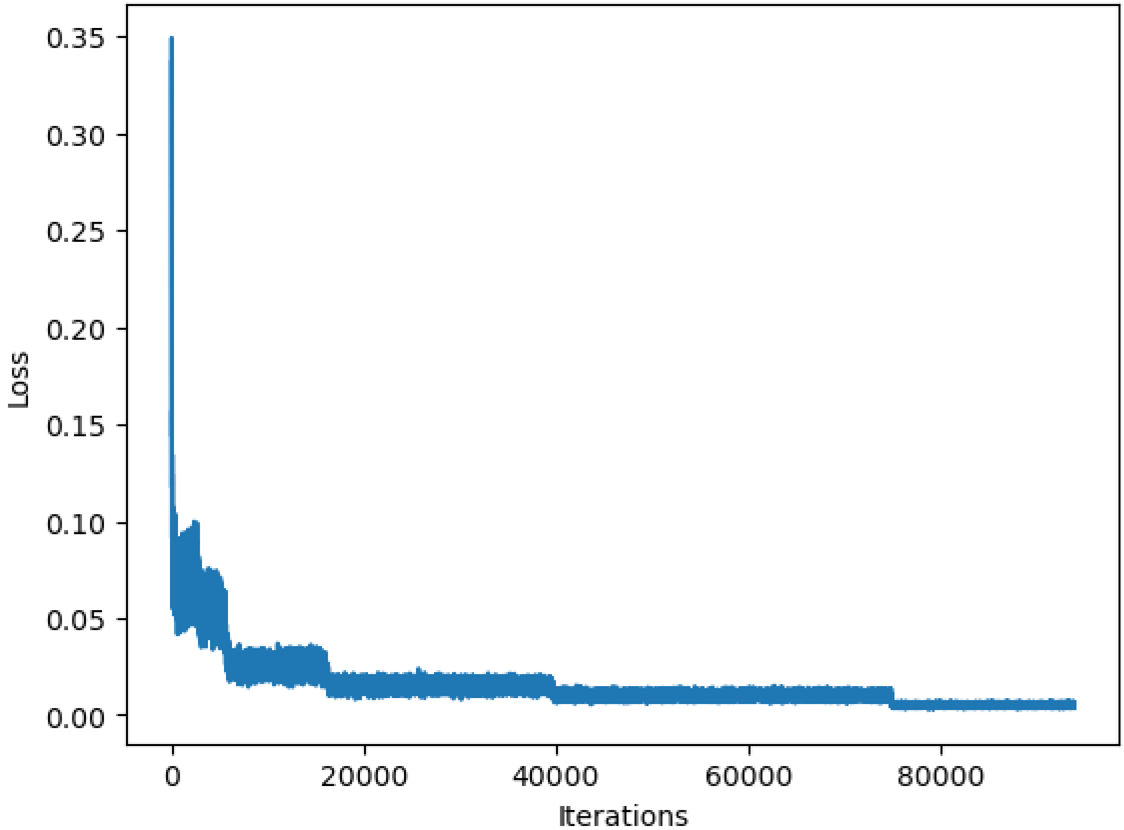
```
epoch [264/300], loss:0.00545817
epoch [265/300], loss:0.00431715
epoch [266/300], loss:0.00551816
epoch [267/300], loss:0.00534865
epoch [268/300], loss:0.00426033
epoch [269/300], loss:0.00526119
epoch [270/300], loss:0.00649111
epoch [271/300], loss:0.00691199
epoch [272/300], loss:0.00636722
epoch [273/300], loss:0.00534541
epoch [274/300], loss:0.00538261
epoch [275/300], loss:0.00602167
epoch [276/300], loss:0.00504890
epoch [277/300], loss:0.00448148
epoch [278/300], loss:0.00694078
epoch [279/300], loss:0.00472259
epoch [280/300], loss:0.00408667
epoch [281/300], loss:0.00430530
epoch [282/300], loss:0.00517836
epoch [283/300], loss:0.00397340
epoch [284/300], loss:0.00656115
epoch [285/300], loss:0.00469791
epoch [286/300], loss:0.00403703
epoch [287/300], loss:0.00502734
epoch [288/300], loss:0.00566571
epoch [289/300], loss:0.00453069
epoch [290/300], loss:0.00460878
epoch [291/300], loss:0.00412072
epoch [292/300], loss:0.00621012
epoch [293/300], loss:0.00533406
epoch [294/300], loss:0.00514319
epoch [295/300], loss:0.00585328
epoch [296/300], loss:0.00399428
epoch [297/300], loss:0.00497398
epoch [298/300], loss:0.00542437
epoch [299/300], loss:0.00350400
epoch [300/300], loss:0.00568663
```

In [36]:

```
l = []
for j in range(len(losses)):
    a = losses[j].detach().numpy()
    l.append(a)

# Defining the Plot Style
plt.plot(l)
plt.style.use('fivethirtyeight')
plt.xlabel('Iterations')
plt.ylabel('Loss')
```

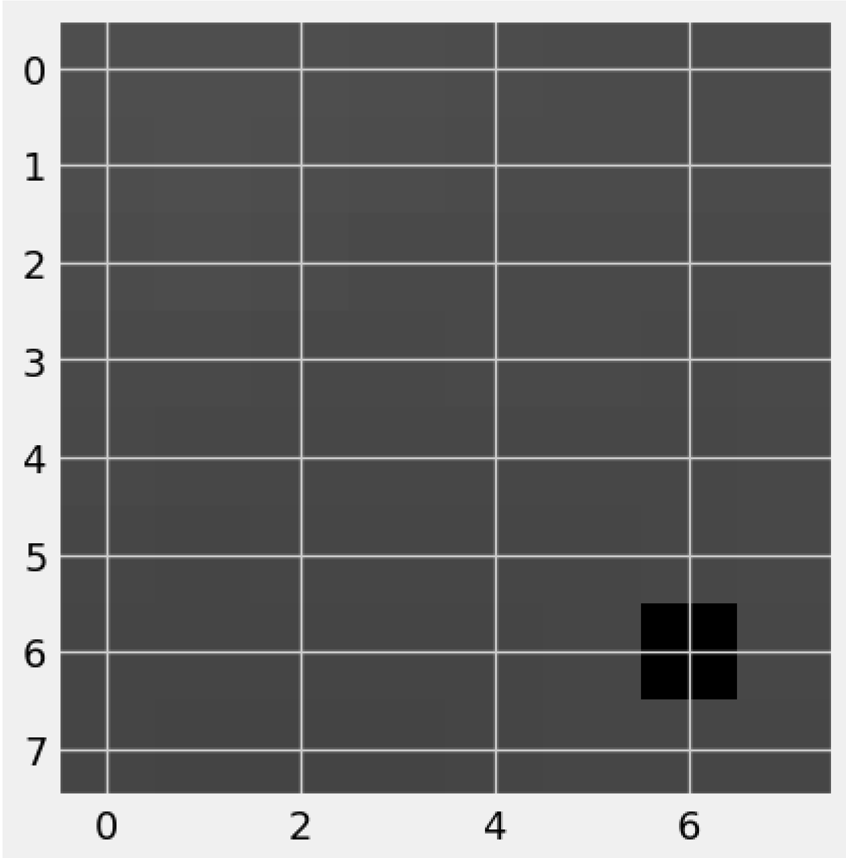
Out[36]:

```
Text(0, 0.5, 'Loss')
```



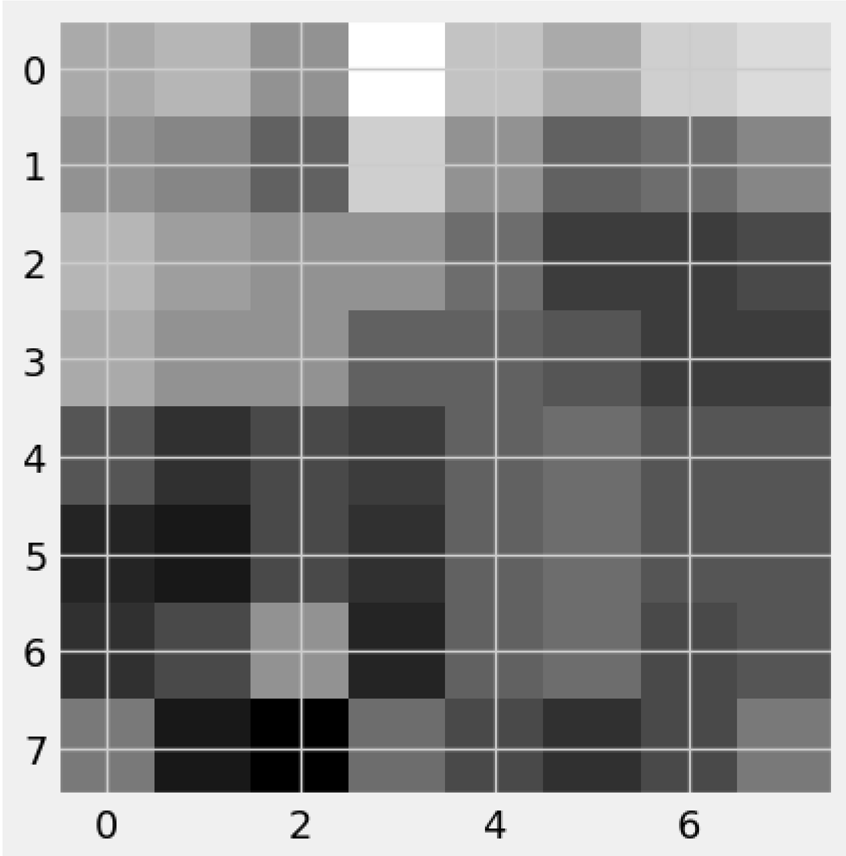
In [44]:

```
# print(reconstructed.shape)
for i, item in enumerate(reconstructed):
    item = item.reshape(-1, k, k)
    plt.imshow(item[0].detach().numpy(),cmap='gray',vmin=0, vmax=1)
#     plt.show()
```



In [41]:

```
for i, item in enumerate(image):
    # Reshape the array for plotting
    item = item.reshape(-1, k, k)
    plt.imshow(item[0],cmap='gray')
```



In [ ]:

In [ ]:

In [ ]: