Guide for DeepL SR-SMLM

by Hanna MANKO

11.10.2021

Contents

# Preparation of the patch sets……………..………..………..………..………..………..………..………..………..………..….….1

# Training the Neural Network……………..………..………..………..………..………..………..………..………..………..….…..1

**Introduction**

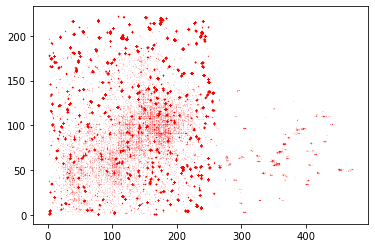
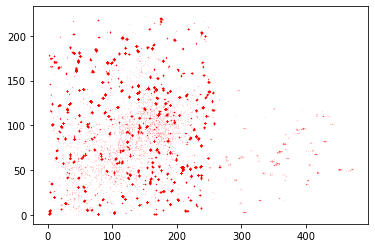
This is a guide that shows how to prepare the training set and train a network for SR-SMLM. The training set contains corresponding low and high SNR images creates from same data.

1. **Preparation of patch sets**
2. ##################### Import the required libraries ###################
3. import numpy as np
4. import pandas as pd
5. from tifffile import imread, imsave
6. from sklearn.cluster import DBSCAN
7. from matplotlib import pyplot as plt
8. from numpy import unique
9. from numpy import where
10. import time
11. import math
12. import random
13. from skimage.morphology import disk
14. from scipy.ndimage.morphology import white\_tophat
15. from joblib import Parallel, delayed
17. ### Defining Patches Size
18. image\_height = 16 ## image height in pixels
19. loc\_image\_width = 16 ## image width in pixels
20. image\_chanels = 1 ##
21. loc\_sp\_distance = 230 ## distance between localization and its spectral peak
22. # measured from original image / in pixels
23. sp\_image\_width = 48 #### spectral image width
24. half\_width = int(loc\_image\_width/2) ##

Next we need to define all the functions we will use later

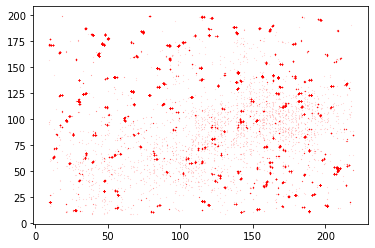
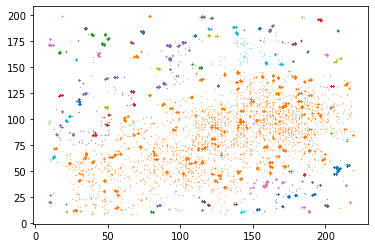
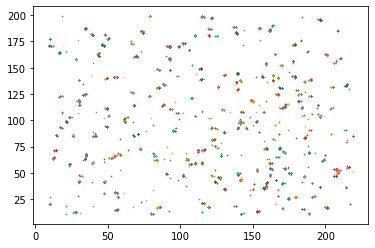
1. ﻿##################### DEFINING THE FUNCTIONS ###################
2. ###
3. ##### ####### Useful function to plot on one figure multiple patches
4. def show\_images(patches):
5. n = int(math.sqrt(len(patches))+2)
6. m = int(math.sqrt(len(patches)))
7. plt.figure(figsize = (50,40))
8. for i in range (0, len(patches)):
9. plt.subplot(n,m,i+1)
10. plt.title(i, fontsize=25)
11. plt.imshow(patches[i])
12. plt.show()
14. #######
15. def set\_coordinates(i, half\_width, loc\_sp\_distance, sp\_image\_width):
16. row = np.array(where(cluster == i))
17. ## coordinates for localization part
18. shift\_y = random.randint(-6,6) ## shift for x and y coordinates. To move the box around localization
19. shift\_x = random.randint(-6,6) ##
20. # shift\_y = 0 ## if shift is not required. When we need to create patches
21. # shift\_x = 0 ##
22. y1 = int(dat[np.array(row)[0,0],0]-half\_width-shift\_y) # start y-coordinate of the box around localization
23. y2 = int(dat[np.array(row)[0,0],0]+half\_width-shift\_y) # end y-coordinate of the box around localization
24. x1 = int(dat[np.array(row)[0,0],1]-half\_width-shift\_x) # start x-coordinate of the box around localization
25. x2 = int(dat[np.array(row)[0,0],1]+half\_width-shift\_x) # end y-coordinate of the box around localization
26. ## coordinates for spectral part
27. y11 = int(dat[np.array(row)[0,0],0]+loc\_sp\_distance-shift\_y) # start y-coordinate of the box around spectra
28. y21 = int(dat[np.array(row)[0,0],0]+loc\_sp\_distance + sp\_image\_width-shift\_y) # end y-coordinate of the box around spectra
29. return(y1,y2,x1,x2,y11,y21,row)
31. #### The code was first oncate to be able to create patches of different types
32. ## Function that returns image shape depending on patch type chosen
33. def image\_type(image\_height, sp\_image\_width,loc\_image\_width):
34. if only\_loc == True: ## to have only localization part
35. im\_shape = (image\_height, loc\_image\_width)
36. if only\_spectra == True: ## to have only spectral part
37. im\_shape = (image\_height, sp\_image\_width)
38. if combined == True: ## to have both localization and spectral parts
39. im\_shape = (image\_height, sp\_image\_width+loc\_image\_width)
40. return (im\_shape)
42. # Function that forms the patches depending on patch type chosen
43. sp\_cor = 0 ### the number of pixels in x-axis to correct position of the box around spectral part
44. def patches\_formation(row, stack):
45. #sp\_cor = random.randint(-2,8) ## to introduce random shift for the box around spectral part
46. frame = dat[:,2][row]
47. if only\_loc == True:
48. patch = np.float64(stack[frame, x1:x2, y1:y2])
49. patch = patch.reshape(row.shape[1],16,16)
50. if only\_spectra == True:
51. patch = np.float64(stack[frame, x1:x2, y11:y21])
52. patch = patch.reshape(row.shape[1],16,48)
53. if combined == True:
54. patch = np.concatenate((np.float64(stack[frame, x1:x2, y1:y2]),
55. np.float64(stack[frame, x1-sp\_cor:x2-sp\_cor, y11:y21])), axis=3)
56. patch = patch.reshape(row.shape[1],16,64)
57. return(patch)
59. ####### Function to calculate Spatial Frecuency as described in (Shutao Li et al.
60. # ‘Combination of images with diverse focuses using the spatial frequency’, Information Fusion
61. # <https://doi>.org/10.1016/S1566-2535(01)00038-0.
62. def SF\_calculator(patch):
63. MN = image\_height\*sp\_image\_width
64. rf = np.sqrt((1/MN)\*np.sum([np.abs((patch[:,n]-patch[:,n-1])\*\*2) for n in range(1,sp\_image\_width-1)]))
65. cf = np.sqrt((1/MN)\*np.sum([np.abs((patch[m,:]-patch[m-1,:])\*\*2) for m in range(1,image\_height-1)]))
66. SF = np.sqrt(rf\*\*2 + cf\*\*2)
67. return SF
69. # Two functions to denoise spectral and localization part
70. selem = disk(10)
71. def d\_spectra(img, tempIm, ampFact = .5): ## ampFact is introduced to increase signal for the spectral part
72. imgMean1 = tempIm[0, 0:img.shape[2]]/(ampFact\*tempIm[0:img.shape[1],0:img.shape[2]].max())
73. imgMean1 = white\_tophat(imgMean1, footprint = selem)
74. return(imgMean1)
75. def d\_loc(img, tempIm):
76. imgMean2 = tempIm[0, 0:img.shape[2]]/tempIm[0:img.shape[1],0:img.shape[2]].max()
77. imgMean2 = white\_tophat(imgMean2, footprint = selem)
78. return(imgMean2)
80. ####### /// Spatial Frequency calculation and image oncatena \\\
81. def SF\_Image(patch, only\_loc=False, only\_spectra=False, combined=False, denoising = True):
82. SF=[]
83. SF\_sum = 0
84. cumIm = np.zeros((patch[1:].shape))
85. for I in range(0,len(patch[1:])):
86. SF.append(SF\_calculator(patch[i]))
87. cumIm = cumIm + patch[i]\*SF[i]
88. SF\_sum = np.sum(SF)
89. tempIm = cumIm/SF\_sum
90. if denoising ==True:
91. if only\_spectra == True:
92. imgMean = d\_spectra(patch[:,:,loc\_image\_width:], tempIm[:,:,loc\_image\_width:])
93. elif only\_loc == True:
94. imgMean = d\_loc(patch[:,:,0:loc\_image\_width], tempIm[:,:,0:loc\_image\_width])
95. elif combined == True:
96. imgMean = np.concatenate((d\_loc(patch[:,:,0:loc\_image\_width],
97. tempIm[:,:,0:loc\_image\_width]),(d\_spectra(patch[:,:,loc\_image\_width:],
98. tempIm[:,:,loc\_image\_width:]))),axis=1)
99. return(imgMean)
100. else:
101. return(tempIm)
102. ####
103. # function that exclude one frame and apply SF\_Image function to all othe frames
104. def SF\_part(n, patch\_x, pat\_s):
105. fr = [\*range(0,len(patch\_x))]
106. fr.pop(n)
107. pat\_s = SF\_Image(patch\_x[((fr)),:,:], combined=True)
108. return (pat\_s) #

Data preparation part:

2. ﻿############################################################### 3
3. ############################################################### 2
4. ############################################################### 1 ...
5. ########################## ## Put the type of images that you want to obtain equal True
6. ###/// Here we will create images containing both spectral and localization parts
7. only\_spectra = False
8. only\_loc = False
9. combined = True
10. ###\\\
11. ##########################
13. # Reading the oncatenate file created in one of the next plugins (line 147, line 158)
14. data = pd.read\_csv(‘/Users/hannamanko/Desktop/diff\_st/My Library\_2/U-net/for figures/50mW\_4\_THSt\_5.0.csv’)
16. #### localization file from ThunderSTORM (Fiji plugin) ###
17. # ////
18. data = data[data[“Intensity [photon]”]>10000] ## you can filter the data by intensity to discard noise localizations
19. markerSize = 0.05
20. plt.plot(data[“x [nm]”], data[“y [nm]” ], ‘o’, markersize=markerSize, color=’red’)
22. datat = pd.concat([pd.DataFrame(data[“x [nm]”]),pd.DataFrame(data[“y [nm]”]),pd.DataFrame(data[“frame”])], axis = 1, ignore\_index=False)
23. datat[‘y [nm]’] = (datat[‘y [nm]’]/21.36752)/5
24. datat[‘x [nm]’] = (datat[‘x [nm]’]/21.36752)/5
25. ### \\\\
27. #### localization file from PeakFit (part of GDSC SMLM2 plugin in Fiji) ###
28. # /////
29. data = data[data[“Signal”]>35000]
30. plt.plot(data[“origX”], data[“origY” ], ‘o’, markersize=markerSize, color=’red’)

Filtered data

Unfiltered data

1. datat = pd.concat([pd.DataFrame(data[“origX”]),pd.DataFrame(data[“origY”]),pd.DataFrame(data[“Frame”])], axis = 1, ignore\_index=False)
2. data\_n = datat.to\_numpy()
3. ### \\\\\
5. data\_n= datat.to\_numpy()
6. d = data\_n[(data\_n[:,0]>8)&(data\_n[:,0]<220)&(data\_n[:,1]>8)&(data\_n[:,1]<200)]
7. plt.plot(d[:,0], d[:,1], ‘o’, markersize=markerSize, color=’red’)
8. #####
9. ##############################################################################
10. ##### Next we cluster the localizations on consecutive frames to create ‘clean’ images from the same localization
11. ## For this we are using DBSCAN
13. cl = DBSCAN(eps=0.5,min\_samples=3) # clustering ## eps need to be choosed depending on the data (see line 203)
14. cluster = cl.fit\_predict(d[:, 0:2]) # table with indexes of all clusters
15. clusters = unique(cluster) # tacking unique indexes of clusters
16. clusters = clusters[clusters>-1] # discarding noise
17. #---
18. def consecutive(data, stepsize=1): # small function that helps to find consecutive frames for cluster
19. return np.split(data, np.where(np.diff(data) != stepsize)[0]+1)
20. #---
21. dat = pd.concat([pd.DataFrame(d), pd.DataFrame(cluster)], axis = 1) # Adding cluster indexes as additional column to our data
22. dat = dat.to\_numpy() # Converting to numpy
24. cluster\_2 = np.int64(np.full((cluster.shape), -1)) # Creation array full of ‘-1’
25. new\_cluster = 0
26. for I in clusters:
27. ar = consecutive(dat[:,2][(where(dat[:,3] == i))]) # I consecutive indexes in data for each cluster
28. for j in ar: # for ezch of the consecutive index sets
29. if len(j) > 8: # if this set is longer then 8
30. for ii in range(0, len(j)):
31. cluster\_2[where((dat[:,2]==j[ii])&(dat[:,3] == i))] = int(new\_cluster) # put new cluster index to cluster\_2
32. print(new\_cluster)
33. new\_cluster = new\_cluster+1 # increase new\_cluster index by 1
35. clusters\_2 = unique(cluster\_2) # again finding unique clusters
36. clusters\_2 = clusters\_2[clusters\_2>-1] # discarding the noise
37. dat[:,3] = cluster\_2 # put cluster indexes as column 3 in data
38. clusters = clusters\_2 # rewrite clusters
39. del cluster\_2, clusters\_2 # deleting additional variables
41. for clust in clusters: ## Using this loop we can plot data by coloring different clusters (helps to choose eps)
42. row\_ix = where(cluster == clust)
43. plt.scatter(dat[row\_ix, 0], dat[row\_ix, 1], s=markerSize)
44. plt.show() ##

Good eps

Example of too big eps

Here is the main part of code that creates patch sets

2. # Reading the original stack of acquired images
3. stack = imread(“/Users/hannamanko/Desktop/diff\_st/My Library\_2/U-net/for figures/50mW\_4\_MMStack\_Pos0.ome-1.tif”)
5. ﻿start\_time = time.time()
6. print(“--- %s seconds ---" ,(time.time() – start\_time)) # printing start time
7. ### Creating the training set
8. ##########
9. im\_shape = image\_type(image\_height, sp\_image\_width,loc\_image\_width) # Reading image shape
10. count = 0
11. print(“Please, wait until the end”)
12. start\_time = time.time()
13. for I in clusters: # for each cluster index
14. y1,y2,x1,x2,y11,y21,row = set\_coordinates(I,half\_width,loc\_sp\_distance, sp\_image\_width) # defining the coordinates of the box
15. patch\_x = np.zeros((np.shape(row[0])[0], im\_shape[0], im\_shape[1])) # creating the empty patch of defined size
16. try:
17. patch = patches\_formation(row, stack) # forming the patches for one cluster (consecutive localizations)
18. except:
19. patch.shape != im\_shape
20. patch\_x = patch
21. pat\_s = np.zeros(patch\_x.shape)
22. SF\_p = []
23. SF\_sum = []
24. pat\_s = Parallel(n\_jobs=5)(delayed(SF\_part)(n, patch\_x, pat\_s) for n in range(0, len(patch\_x))) # creating ‘clean’ images
25. pat\_s = np.array(pat\_s).reshape(patch\_x.shape) # reshaping
26. SF\_sum = [SF\_calculator(pat\_s[n]) for n in range(0, len(pat\_s))] # saving calculated SF value
27. SF\_p = [SF\_calculator(patch\_x[n]) for n in range(0, len(patch\_x))] # calculating SF value for one excluded patch
28. ind = where(np.array(SF\_sum)/np.array(SF\_p) > ((np.array(SF\_sum)/np.array(SF\_p)).max()- #
29. (np.array(SF\_sum)/np.array(SF\_p)).min())/2) #
30. patch\_ = patch\_x[ind]
31. patch\_sum = pat\_s[ind]
32. if count == 0: # gathering all the patches in one big set
33. patch\_\_ = patch\_ # set of original images
34. patch\_sum\_ = patch\_sum # set of created ‘clean’ images
35. else: #
36. patch\_\_ = np.concatenate((patch\_\_, patch\_), axis=0) # set of original images
37. patch\_sum\_ = np.concatenate((patch\_sum\_, patch\_sum), axis=0) # set of created ‘clean’ images
38. count = count+1
39. print(count)
40. print(“--- %s seconds ---" % (time.time() – start\_time)) # printing end time
42. # Now we can save created raw and ‘clean’ sets of patches
43. imsave(‘path/patch\_sum.tif’, patch\_sum\_)
44. imsave(‘path/patch.tif’, patch\_\_)

In [9]: ….

--- %s seconds --- 0.0

Please, wait until the end

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

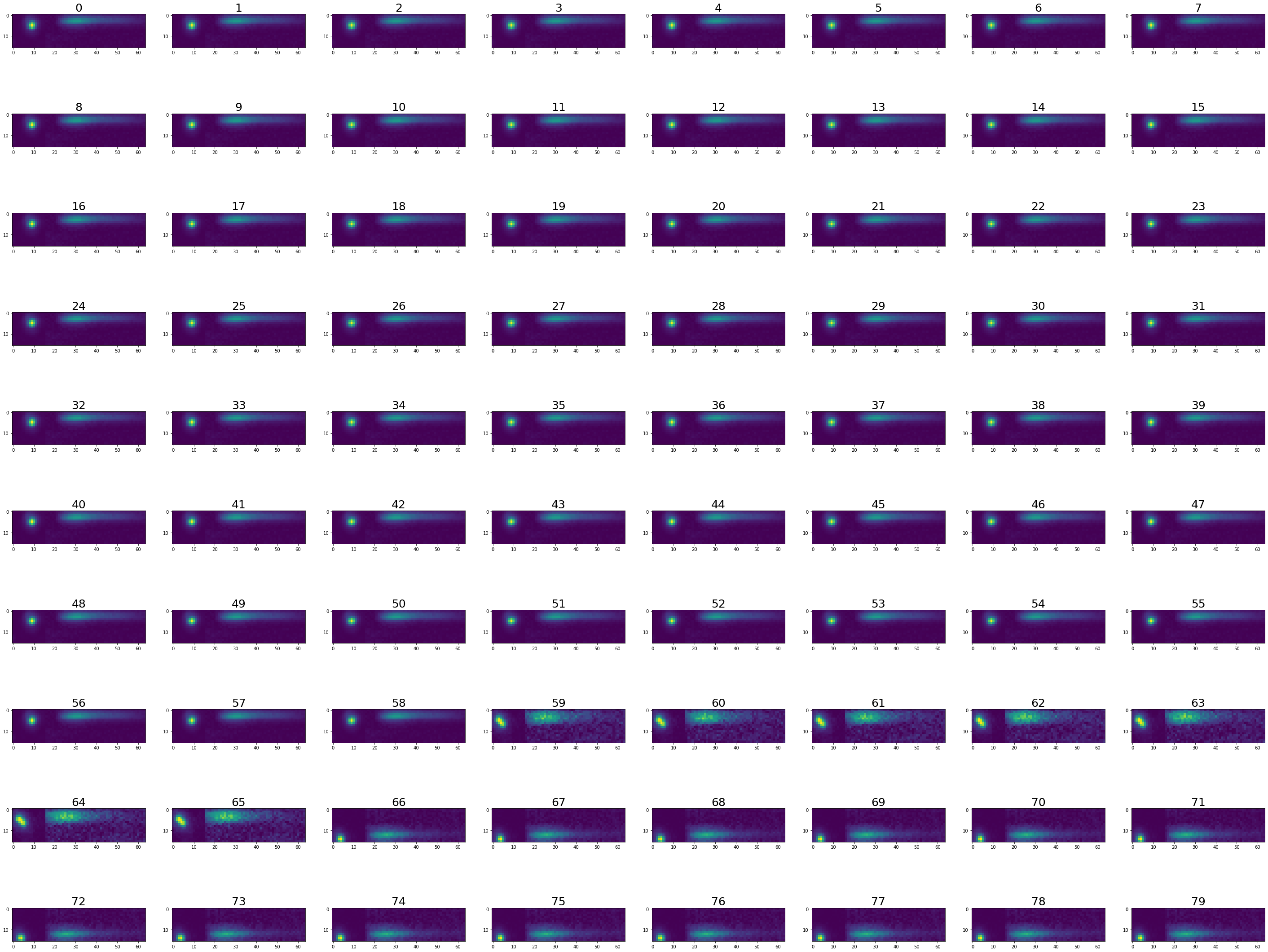
...

This process can take a significant amount of time so it is better to use more powerful computer

In [10]: show\_images(patch\_\_[60:140]) ## building the figure with some of the patches with raw images



In [11]: show\_images(patch\_\_[60:140]) ## building the figure with some of the patches with created ‘clean’ images

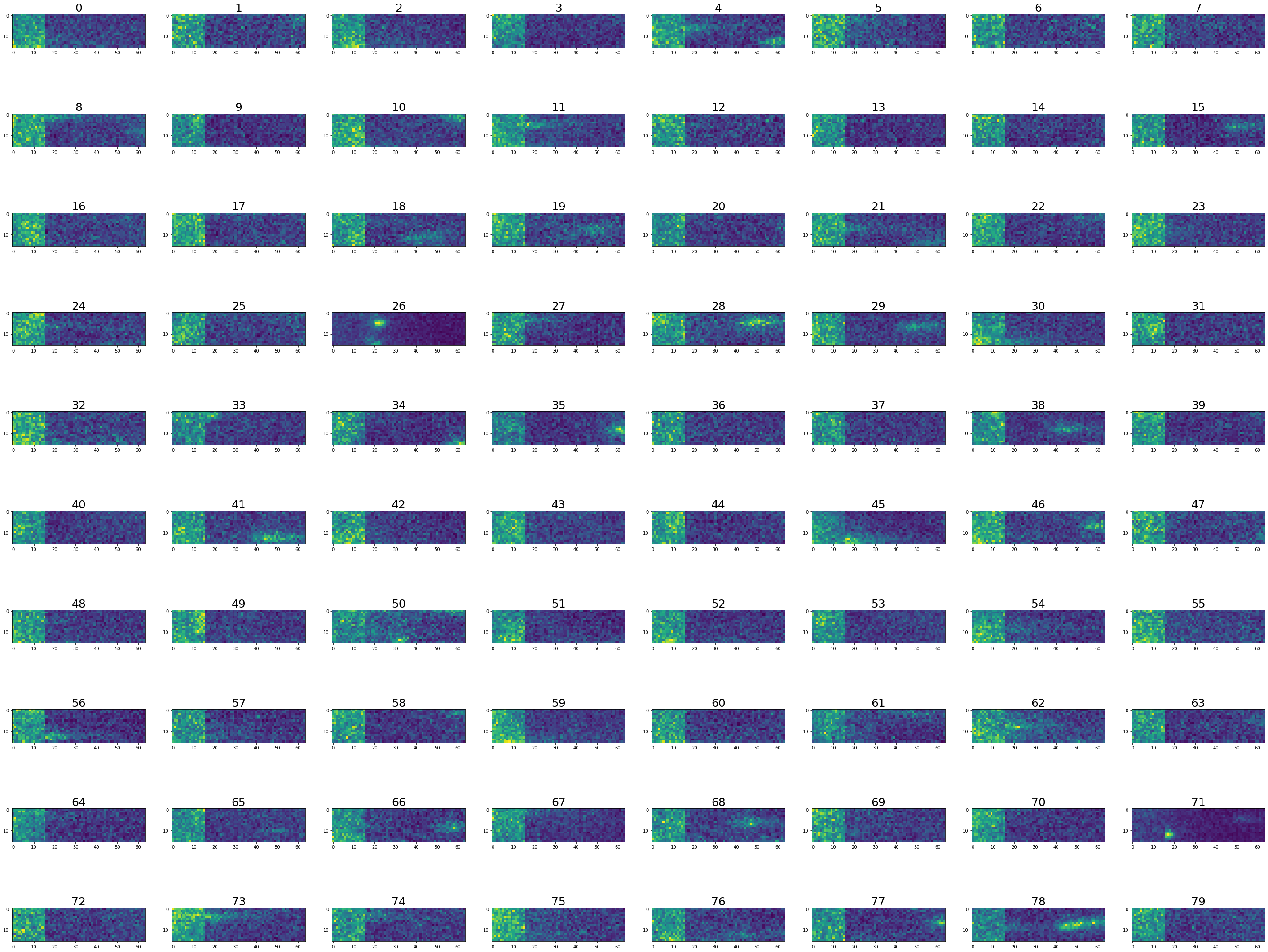


this part is to create noisy patches which are required to properly train the network

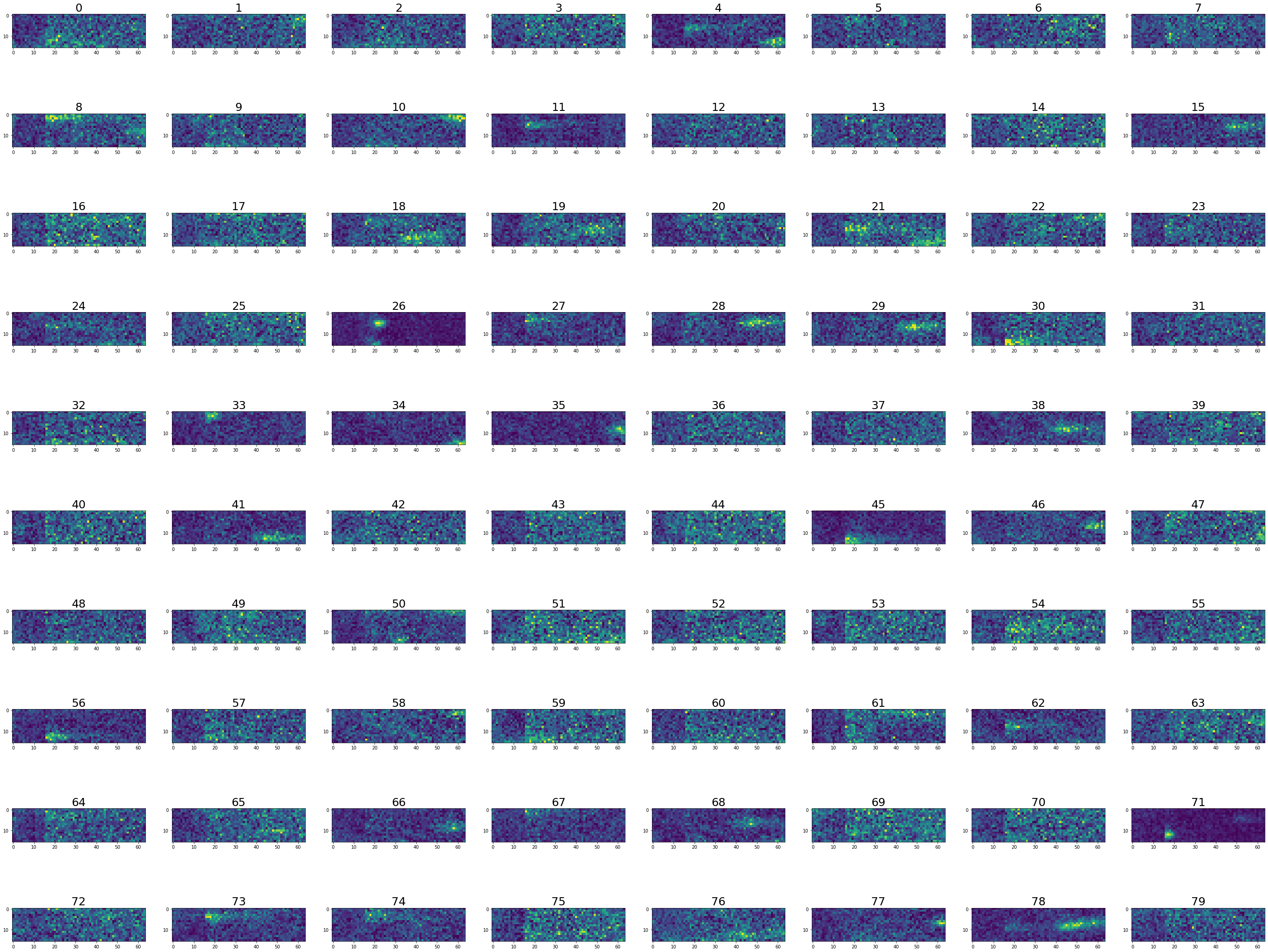
1. ﻿#################################
2. ######### to create noisy patches
3. #################################
4. # We can take any of the stacks used to create patches for the training set
5. stack = imread(‘D:/1/Data/GattaQuant/Gqaunt\_ Fluoro150mW\_sPAINT\_\_11/Gqaunt\_ Fluoro150mW\_sPAINT\_\_11\_MMStack\_Pos0.ome.tif’)
6. stack = stack/60000
7. ######### To create big set of noisy patches this process need to be repeated several times
8. ######### and obtained sets need to be concatenated
9. ### :::::::::
10. im\_shape = image\_type(image\_height, sp\_image\_width,loc\_image\_width)
11. count = 0
12. for I in range(0, len(stack)):
13. for \_ in range(0, 40):
14. rand\_x = random.randint(13,215)
15. rand\_y = random.randint(10,220)
16. y1 = rand\_y-half\_width
17. y2 = rand\_y+half\_width
18. x1 = rand\_x-half\_width
19. x2 = rand\_x+half\_width
20. ## coordinates for spectral part
21. y11 = rand\_y+loc\_sp\_distance
22. y21 = rand\_y+loc\_sp\_distance + sp\_image\_height
23. patch = np.concatenate((np.float64(stack[I, x1:x2, y1:y2]),
24. np.float64(stack[I, x1:x2, y11:y21])), axis=1)
25. try:
26. patch = patch
27. except:
28. patch.shape != im\_shape
29. pat\_s = SF\_Image(np.concatenate((patch.reshape(1, 16, 64), patch.reshape(1, 16, 64)), axis = 0),combined=True)
30. patch\_ = np.concatenate((patch.reshape(1, 16, 64), patch.reshape(1, 16, 64)), axis = 0)
31. patch\_sum = pat\_s
32. if count == 0:
33. patch\_\_ = patch\_[0].reshape(1,16,64)
34. patch\_sum\_ = patch\_sum.reshape(1,16,64)
35. else:
36. patch\_\_ = np.concatenate((patch\_\_, patch\_[0].reshape(1,16,64)), axis=0)
37. patch\_sum\_ = np.concatenate((patch\_sum\_, patch\_sum.reshape(1,16,64)), axis=0)
38. count = count+1
39. print(count)
41. patch\_\_ = imread(‘D:/My Library\_2/U-net/For\_Network/xx/Noise3.tif’)
42. patch\_sum\_ = imread(‘D:/My Library\_2/U-net/For\_Network/yy/Noise3.tif’)

45. # Filtering and deleting the patches that randomly got localization
46. i\_list = []
47. for I in range(0, len(patch\_sum\_)):
48. if patch\_sum\_[I,:,:16].max() > 0.3:
49. i\_list.append(i)
50. patch\_sum\_ = np.delete(patch\_sum\_, [i\_list], axis=0)
51. patch\_\_ = np.delete(patch\_\_, [i\_list], axis=0)

In [12]: show\_images(patch\_\_[60:140]) ## building the figure with some of the ‘noisy’ patches with ‘raw’ images



In [13]: show\_images(patch\_sum\_[60:140]) ## building the figure with some of the ‘noisy’ patches with created ‘clean’ images

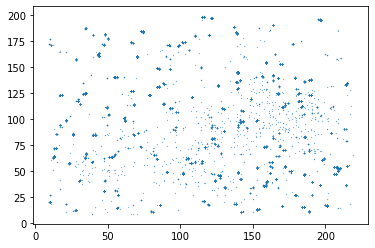


1. ### To concatenate several sets of patches:
2. p = patch\_\_ # writing set of raw patches to new variable
3. ps = patch\_sum\_ # writing set of ‘clean’ patches to new variable
4. ## Sometimes to obtain required number of images in patch set
5. patch\_\_ = np.concatenate((patch\_\_, p), axis = 0) # gathering all the patch sets
6. patch\_sum\_ = np.concatenate((patch\_sum, ps), axis=0)
8. ### :::::::::
9. #########
10. ﻿##############################

This part usually used after we have a trained network to treat new data

1. ##############################
2. ### Creating the same type of patches but only the ‘raw’, so basically we just cut the images in the correct form.
3. ### These patches are created to be treated with our model and get images with high SNR at the output
4. #
5. ####################### !!!! In this part the shift of the box is not used anymore,
6. # so all the variables that induce
7. # shift in the ‘Defining functions’ part need to be disabled
8. ### + for this part the data should not be clustered so we use array **d created in line 167** !!!! ########
10. ##############################
11. x1\_list = [] # list to save original x-coordinates
12. y1\_list = [] # list to save original y-coordinates
13. count = 0
14. for I in range(0,len(d)):
15. y1 = int(d[i][:1][0])-half\_width
16. y2 = y1+2\*half\_width
17. x1 = int(d[i][1:2][0])-half\_width
18. x2 = x1+2\*half\_width
19. ## coordinates for spectral part
20. y11 = int(d[i][:1][0])+loc\_sp\_distance
21. y21 = int(d[i][:1][0])+loc\_sp\_distance + sp\_image\_width
22. try:
23. patch = np.concatenate(((stack[int(d[i][2]), x1:x2, y1:y2]),
24. (stack[int(d[i][2]), x1:x2, y11:y21])), axis=1)
25. if (patch[8,1]+patch[8,8]+patch[8,15]) == 0:
26. continue
27. y1\_list.append(d[i][1:2][0])
28. x1\_list.append(d[i][:1][0])
29. patch = patch.reshape(1,16,64)
30. except:
31. continue
32. if count == 0:
33. patch\_x = patch
34. else:
35. patch\_x = np.concatenate((patch\_x, patch), axis=0)
36. count = count+1
37. print(count)

40. coordinates = np.zeros((len(x1\_list),2)) # creating array
41. coordinates[:,0]= x1\_list # adding x coordinates to 0 column
42. coordinates[:,1]= y1\_list # adding y coordinates to 1st column
44. plt.scatter(coordinates[:,0],coordinates[:,1],s=markerSize,cmap=plt.cm.jet) # we can plot the coordinates to check if everything is ok

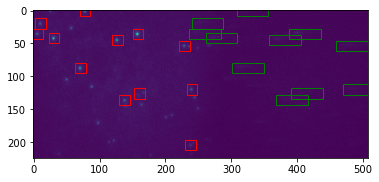


In [14]: show\_images(patch\_x[140:200]) ## building the figure with some of the patches ()



1. # Saving the original coordinates as \*.csv file and created patch as \*.tif
2. pd.DataFrame(coordinates).to\_csv(‘/Users/hannamanko/Desktop/diff\_st/My Library\_2/U-net/for figures/coordinates\_5.0.csv’,index = False)
3. imsave(‘/Users/hannamanko/Desktop/diff\_st/My Library\_2/U-net/for figures/50mW\_4\_patch\_m5.0.tif’, patch\_x)
4. ##############################

Small additional part to draw rectangles around localizations and corresponding spectra

1. ﻿###################################################
2. ################# To draw rectangles
3. ###################################################
4. from matplotlib.patches import Rectangle
5. dd = stack[980] # Choosing one frame from the images stack
6. plt.imshow(dd) # showing this image
7. datt = data.iloc[where(data[‘Frame’] == 980)] # choosing localizations only from this frame
8. datt = datt[datt[‘origX’]<250]
9. datt = datt[datt[‘Signal’]>60000]  # Filtering by intensity
11. cord = pd.concat((datt[‘origX’], datt[‘origY’]), axis = 1, ignore\_index=False)plt.scatter(loc[‘origX’], loc[‘origY’])
13. ax = plt.gca()
15. rect = Rectangle((x1,y1),16,16,linewidth=1,edgecolor=’r’,facecolor=’none’) # Creating a Rectangle
17. ax.add\_patch(rect)
19. for I in range(0, len(cord)):
20. y1 = cord.iloc[I,1]-half\_width
21. x1 = cord.iloc[I,0]-half\_width
22. y11 = cord.iloc[I,0]+loc\_sp\_distance
23. plt.imshow(dd)
24. plt.gca().add\_patch(Rectangle((x1,y1),16,16,linewidth=1,edgecolor=’r’,facecolor=’none’))
25. plt.gca().add\_patch(Rectangle((y11,y1),48,16,linewidth=1,edgecolor=’green’,facecolor=’none’))
26. plt.show()

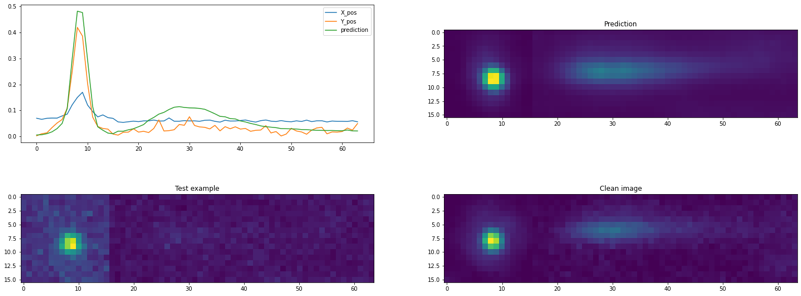
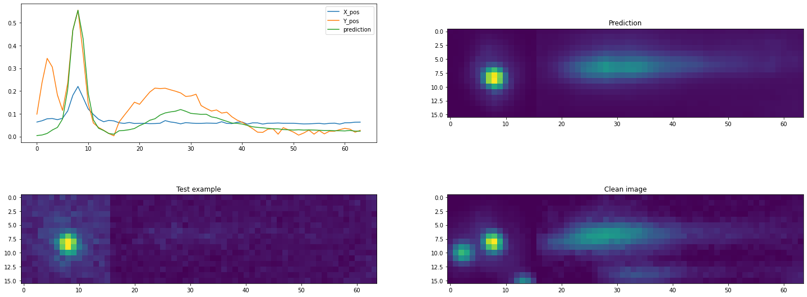
All the created images can be saved as .tif stacks then it is possible to open it using Fiji if it is required. The other possibility is to save it as \*.csv table, this way it is faster to upload the images in python.

Here you can see first way, when stacks were created separately so it required to concatenate everything into one big stack:

1. ﻿##############################################
2. ######## Downloading image stacks ####
4. X\_train\_25\_2 = imread('D:/My Library\_2/U-net/For\_Network/x\_new/25\_2\_patch\_tr.tif')
5. X\_train\_50\_1 = imread('D:/My Library\_2/U-net/For\_Network/x\_new/50\_1\_patch\_tr.tif')
6. X\_train\_50\_2 = imread('D:/My Library\_2/U-net/For\_Network/x\_new/50\_2\_patch\_tr.tif')
7. X\_train\_75\_1 = imread('D:/My Library\_2/U-net/For\_Network/x\_new/75\_1\_patch\_tr.tif')
8. X\_train\_75\_2 = imread('D:/My Library\_2/U-net/For\_Network/x\_new/75\_2\_patch\_tr.tif')
9. X\_train\_100\_1 = imread('D:/My Library\_2/U-net/For\_Network/x\_new/100\_1\_patch\_tr.tif')
10. X\_train\_100\_2 = imread('D:/My Library\_2/U-net/For\_Network/x\_new/100\_2\_patch\_tr.tif')
11. X\_train\_125\_2 = imread('D:/My Library\_2/U-net/For\_Network/x\_new/125\_2\_patch\_tr.tif')
12. X\_train\_125\_3 = imread('D:/My Library\_2/U-net/For\_Network/x\_new/125\_3\_patch\_tr.tif')
13. X\_train\_150\_1 = imread('D:/My Library\_2/U-net/For\_Network/x\_new/150\_1\_patch\_tr.tif')
14. X\_train\_150\_2 = imread('D:/My Library\_2/U-net/For\_Network/x\_new/150\_2\_patch\_tr.tif')
15. X\_noise = imread('D:/My Library\_2/U-net/For\_Network/xx/Noise.tif')
16. X\_noise2 = imread('D:/My Library\_2/U-net/For\_Network/xx/Noise2.tif')
17. X\_noise3 = imread('D:/My Library\_2/U-net/For\_Network/xx/Noise3.tif')
19. X\_train = np.concatenate((X\_noise/1.5, X\_train\_25\_2,X\_train\_50\_1,X\_train\_50\_2,
20. X\_train\_75\_1,X\_train\_75\_2,X\_train\_100\_1,
21. X\_train\_125\_2,X\_noise2/1.5,X\_train\_125\_3,X\_train\_150\_1,X\_train\_150\_2,X\_noise3/1.5 ), axis=0)
23. del X\_train\_25\_2,X\_train\_50\_1,X\_train\_50\_2
24. del X\_train\_75\_1,X\_train\_75\_2,X\_train\_100\_1,X\_train\_100\_2
25. del X\_train\_125\_2,X\_train\_125\_3,X\_train\_150\_1,X\_train\_150\_2, X\_noise, X\_noise2, X\_noise3
27. X\_train = X\_train/60000 # normalisation
29. X\_train = X\_train.reshape((-1, image\_height, image\_width, 1))
30. ###//////////////
31. Y\_train\_25\_2 = imread('D:/My Library\_2/U-net/For\_Network/y\_new/25\_2\_patch\_sum.tif')
32. Y\_train\_50\_1 = imread('D:/My Library\_2/U-net/For\_Network/y\_new/50\_1\_patch\_sum.tif')
33. Y\_train\_50\_2 = imread('D:/My Library\_2/U-net/For\_Network/y\_new/50\_2\_patch\_sum.tif')
34. Y\_train\_75\_1 = imread('D:/My Library\_2/U-net/For\_Network/y\_new/75\_1\_patch\_sum.tif')
35. Y\_train\_75\_2 = imread('D:/My Library\_2/U-net/For\_Network/y\_new/75\_2\_patch\_sum.tif')
36. Y\_train\_100\_1 = imread('D:/My Library\_2/U-net/For\_Network/y\_new/100\_1\_patch\_sum.tif')
37. Y\_train\_100\_2 = imread('D:/My Library\_2/U-net/For\_Network/y\_new/100\_2\_patch\_sum.tif')
38. Y\_train\_125\_2 = imread('D:/My Library\_2/U-net/For\_Network/y\_new/125\_2\_patch\_sum.tif')
39. Y\_train\_125\_3 = imread('D:/My Library\_2/U-net/For\_Network/y\_new/125\_3\_patch\_sum.tif')
40. Y\_train\_150\_1 = imread('D:/My Library\_2/U-net/For\_Network/y\_new/150\_1\_patch\_sum.tif')
41. Y\_train\_150\_2 = imread('D:/My Library\_2/U-net/For\_Network/y\_new/150\_2\_patch\_sum.tif')
42. Y\_noise = imread('D:/My Library\_2/U-net/For\_Network/yy/Noise.tif')
43. Y\_noise2 = imread('D:/My Library\_2/U-net/For\_Network/yy/Noise2.tif')
44. Y\_noise3 = imread('D:/My Library\_2/U-net/For\_Network/yy/Noise3.tif')
46. Y\_train = np.concatenate((Y\_noise/1.5, Y\_train\_25\_2, Y\_train\_50\_1,Y\_train\_50\_2,
47. Y\_train\_75\_1,Y\_train\_75\_2,Y\_train\_100\_1,
48. Y\_train\_125\_2,Y\_noise2/1.5,Y\_train\_125\_3,Y\_train\_150\_1, Y\_train\_150\_2, Y\_noise3/1.5), axis=0)
50. del Y\_train\_25\_2,Y\_train\_50\_1,Y\_train\_50\_2
51. del Y\_train\_75\_1,Y\_train\_75\_2,Y\_train\_100\_1,Y\_train\_100\_2
52. del Y\_train\_125\_2,Y\_train\_125\_3,Y\_train\_150\_1,Y\_train\_150\_2, Y\_noise, Y\_noise2, Y\_noise3
54. Y\_train = Y\_train/Y\_train.max()
55. Y\_train = Y\_train.reshape(-1, image\_height, image\_width, 1)
56. ﻿###. And we can save final stacks for training set
57. imsave('C:/Users/hmanko/Desktop/X\_train.tif', X\_train)
58. imsave('C:/Users/hmanko/Desktop/Y\_train.tif', Y\_train)

1. **Training the Neural Network**
2. **﻿**import keras
3. import tensorflow as tf
4. import os
6. import numpy as np
7. from matplotlib import pyplot as plt
8. import random
9. from tifffile import imread, imsave
10. from PIL import Image

13. from skimage import io, img\_as\_ubyte
14. from skimage.transform import resize, rescale
15. import random
16. import pandas as pd
17. from tqdm import tqdm
19. from tensorflow.keras.layers import Lambda,Input,Conv2D,BatchNormalization,AveragePooling2D,LeakyReLU,Conv2DTranspose,concatenate,UpSampling2D,Dropout
20. from skimage.morphology import disk, white\_tophat
22. from numpy import expand\_dims
23. from keras.preprocessing.image import ImageDataGenerator

26. image\_width = 64
27. image\_height = 16
28. image\_chanels = 1
30. ﻿
31. X\_train = imread('C:/Users/hmanko/Desktop/X\_train.tif')
32. Y\_train = imread('C:/Users/hmanko/Desktop/Y\_train.tif')
34. inputs = Input((image\_height, image\_width, image\_chanels))
36. c0=Conv2D(16, (6,6),activation = 'elu',kernel\_initializer='he\_normal', padding = 'same')(inputs)
37. c0=BatchNormalization(axis=-1)(c0)
38. c0=LeakyReLU(alpha=0.2)(c0)
39. c0=Conv2D(16, (6,6),activation = 'elu',strides = 2,kernel\_initializer='he\_normal', padding = 'same')(c0)
40. c0=BatchNormalization(axis=-1)(c0)
41. c0=LeakyReLU(alpha=0.2)(c0)
43. c1=Conv2D(32, (6,6),activation = 'elu',kernel\_initializer='he\_normal', padding = 'same')(c0)
44. c1=BatchNormalization(axis=-1)(c1)
45. c1=LeakyReLU(alpha=0.2)(c1)
46. c1=Conv2D(32, (6,6),activation = 'elu',strides = 2, kernel\_initializer='he\_normal', padding = 'same')(c1)
47. c1=BatchNormalization(axis=-1)(c1)
48. c1=LeakyReLU(alpha=0.2)(c1)
50. c2=Conv2D(64, (6,6),activation = 'elu',kernel\_initializer='he\_normal', padding = 'same')(c1)
51. c2=BatchNormalization(axis=-1)(c2)
52. c2=LeakyReLU(alpha=0.2)(c2)
53. c2=Conv2D(64, (6,6),activation = 'elu',strides = 2,kernel\_initializer='he\_normal',padding = 'same')(c2)
54. c2=BatchNormalization(axis=-1)(c2)
55. c2=LeakyReLU(alpha=0.2)(c2)
57. c3=Conv2D(128, (6,6),activation = 'elu',kernel\_initializer='he\_normal', padding = 'same')(c2)
58. c3=BatchNormalization(axis=-1)(c3)
59. c3=LeakyReLU(alpha=0.3)(c3)
61. u4=Conv2DTranspose(64, (6,6), padding ='same')(c3)
62. u4=concatenate([u4,c2])
63. c4=UpSampling2D(size=2)(u4)
64. c4=Conv2D(64, (6,6),activation = 'elu',kernel\_initializer='he\_normal', padding = 'same')(c4)
65. c4=BatchNormalization(axis=-1)(c4)
66. c4=LeakyReLU(alpha=0.2)(c4)
68. u5=Conv2DTranspose(32, (6,6), padding ='same')(c4)
69. u5=concatenate([u5,c1])
70. c5=UpSampling2D(size=2)(u5)
71. c5=Conv2D(32, (6,6),activation = 'elu',kernel\_initializer='he\_normal', padding = 'same')(c5)
72. c5=BatchNormalization(axis=-1)(c5)
73. c5=LeakyReLU(alpha=0.2)(c5)
75. u6=Conv2DTranspose(16, (6,6), padding ='same')(c5)
76. u6=concatenate([u6,c0])
77. c6=UpSampling2D(size=2)(u6)
78. c6=Conv2D(16, (6,6),activation = 'elu',kernel\_initializer='he\_normal', padding = 'same')(c6)
79. c6=BatchNormalization(axis=-1)(c6)
80. c6=LeakyReLU(alpha=0.2)(c6)
82. outputs = Conv2D(1, (1,1), activation ='relu')(c6)
84. model = tf.keras.Model(inputs = [inputs], outputs = [outputs])
85. model.compile(optimizer = 'rmsprop', loss="mean\_squared\_error")
86. model.summary()
88. reduce\_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val\_loss', factor=0.2,patience=4, min\_lr=0.001)
89. earlyStop = keras.callbacks.EarlyStopping(patience=10, verbose=1, restore\_best\_weights=True)
90. callbacks\_list = [earlyStop, reduce\_lr]
92. history = model.fit(X\_train,Y\_train,validation\_split=0.1, batch\_size=80, epochs=40)
94. model.save("D:/My Library\_2/model\_mix\_30.07\_19\_BS\_50(6,6)\_.h5") # saving the model
95. ﻿
96. #############################################
97. ## Downloading the test patches
98. X\_test = imread('/path/\*.tif')
99. X\_test = X\_test/60000
100. X\_test = X\_test.reshape(-1, image\_height, image\_width, 1)
102. Y\_test = imread('/path/\*.tif')
103. Y\_test = Y\_test/60000
104. Y\_test = Y\_test.reshape(-1, image\_height, image\_width, 1)
106. prediction = model.predict(X\_test)
108. ######################
109. num = random.randint(1,len(X\_test))
110. plt.figure(figsize = (25,10))
111. plt.subplot(221)
112. plt.plot(list(range(0,64)),X\_test[num,10,:,0])
113. plt.plot(list(range(0,64)),Y\_test[num, 8,:,0])
114. plt.plot(list(range(0,64)),prediction[num, 10,:,0])
115. plt.legend(['X\_pos', 'Y\_pos','prediction'])
116. plt.subplot(222)
117. plt.imshow(prediction[num,:,:,0])
118. plt.title("Prediction")
119. plt.subplot(223)
120. plt.imshow(X\_test[num,:,:,0])
121. plt.title("Test example")
122. plt.subplot(224)
123. plt.imshow(Y\_test[num,:,:,0])
124. plt.title("Clean image")
125. plt.show()

Here, because ‘clean’ image is created using multiple images of the same localization we can see other localizations appearing on the image, that do not correspond to raw images.

2. imsave('D:/1/Data/GataQuand\_\_15-150mW/50\_4\_pred.tif', prediction)
3. ﻿
5. ## To download already trained model
6. model = load\_model("D:/My Library\_2/model\_mix\_19.08\_25\_BS\_80(6,6)\_.h5")