U2288061

September 26, 2023

[1]:
```
!pip install tqdm
!pip install tabulate
!pip install scipy
!pip install torch
!pip install torchvision
```

```
Requirement already satisfied: tqdm in
c:\users\cynthia\anaconda3\envs\pytorch\lib\site-packages (4.65.0)
Requirement already satisfied: colorama in
c:\users\cynthia\anaconda3\envs\pytorch\lib\site-packages (from tqdm) (0.4.6)
```

[1]:
```
import numpy as np
import pandas as pd
from pandas.core.frame import DataFrame
import matplotlib.pyplot as plt
from skimage import data
from skimage.color import rgb2hed, hed2rgb
from sklearn.decomposition import PCA
from skimage.feature import greycomatrix, greycoprops
from skimage.color import rgb2gray
from skimage.util import img_as_ubyte
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import GridSearchCV
from sklearn import*
import warnings
warnings.filterwarnings("ignore")
from tabulate import tabulate
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.svm import LinearSVR
from sklearn.preprocessing import StandardScaler
import torch
import torch.nn as nn
import torchvision
```

```
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
from tqdm import tqdm
from scipy import stats
from scipy.stats import pearsonr, spearmanr
```

[2]:
```
X = np.load("images.npy")#read images
Y = pd.read_csv('counts.csv')#read cell counts
F = np.loadtxt('split.txt')#read fold information
```

1 Question No. 1: (Data Analysis)

1.0.1 i. How many examples are there in each fold?

[5]:
```
value,count=np.unique(F,return_counts=True)
# print('There are', fold_sizes, 'in each fold')
for i in range(len(value)):
    print('There are', count[i], 'examples in fold',int(value[i]),'.')
```

```
There are 1622 examples in fold 1 .
There are 1751 examples in fold 2 .
There are 1608 examples in fold 3 .
```

1.0.2 ii. Show some image examples using plt.imshow. Describe your observations on what you see in the images and how it correlates with the cell counts of different types of cells and the overall number of cells.

From the randomly generated images, I have observed distinct differences in cell morphology, size, and staining between different types of cells. For instance, in example 2162, connective cells exhibit a longer morphology, whereas lymphocytes and eosinophils are rounder. Furthermore, different types of cell staining can be used to highlight specific cell components, with lymphocytes and eosinophils staining darker than connective cells in the same sample.

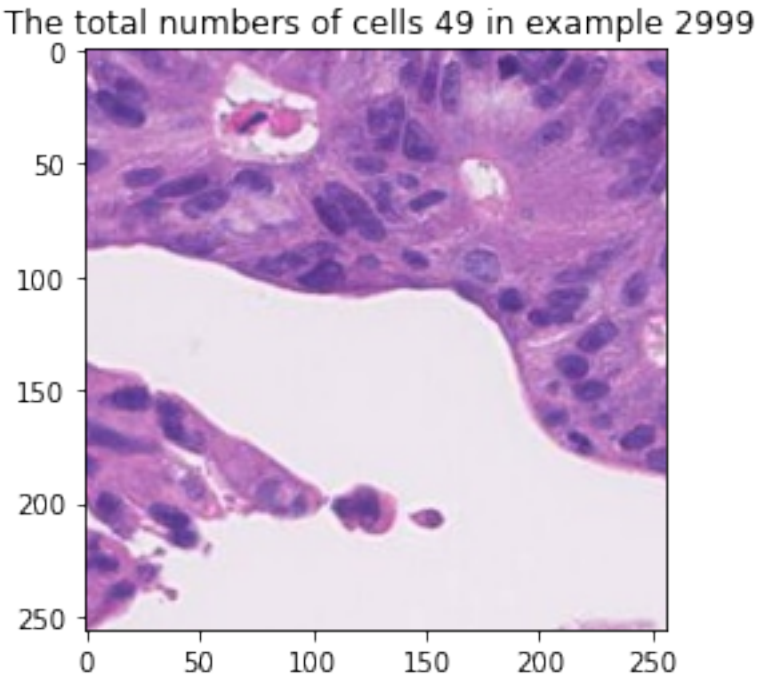
By analyzing the cell counts of different types of cells and the overall number of cells, I have noticed that the density and size of cells can vary significantly between different cell types. By analyzing the number of cells in a given area, it is possible to estimate the cell density and identify any differences in cell distribution. Additionally, by measuring the size of cells, one can determine the average size of the cells and look for variations in size between different cell types. Overall, these observations allow for a more accurate identification and characterization of different types of cells, which is crucial for medical research, pathology, and cell biology.

Images with a higher total cell count tend to exhibit more diversity in cell types. The count of eosinophil cells is not correlated with the total cell count, as it remains constant regardless of changes in the total count. However, an increase in the count of each cell type will result in an increase in the total cell count.

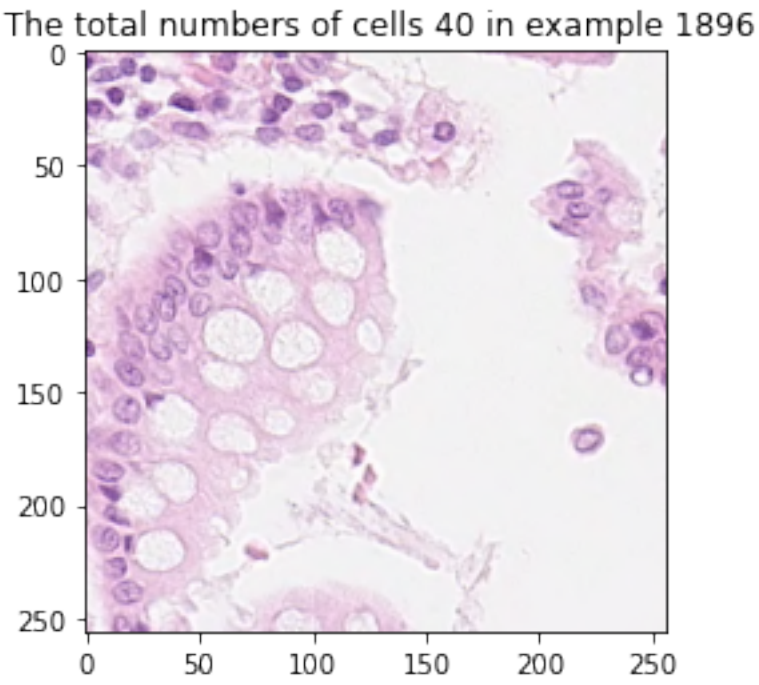
There are 20 examples. Each image displays the count of each cell type above it, while the total cell count for each image is included in the image title.

[4]:
```
# randomly pick 20 examples
idx = np.random.randint(X.shape[0], size=20)
idx
for i in idx:
    plt.imshow(X[i])
    print('Type'+str(Y.iloc[i][Y.iloc[i]>0]))

    plt.title(' The total numbers of cells '+str(sum(Y.iloc[i]))+' in example␣
 ↪'+str(i))
    plt.show()
```

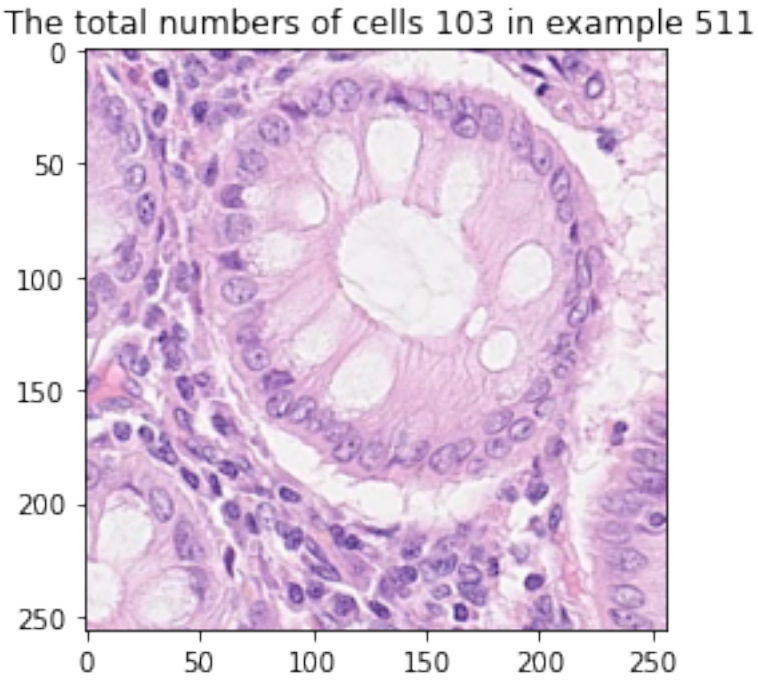
```
Typeepithelial    49
Name: 2999, dtype: int64
```



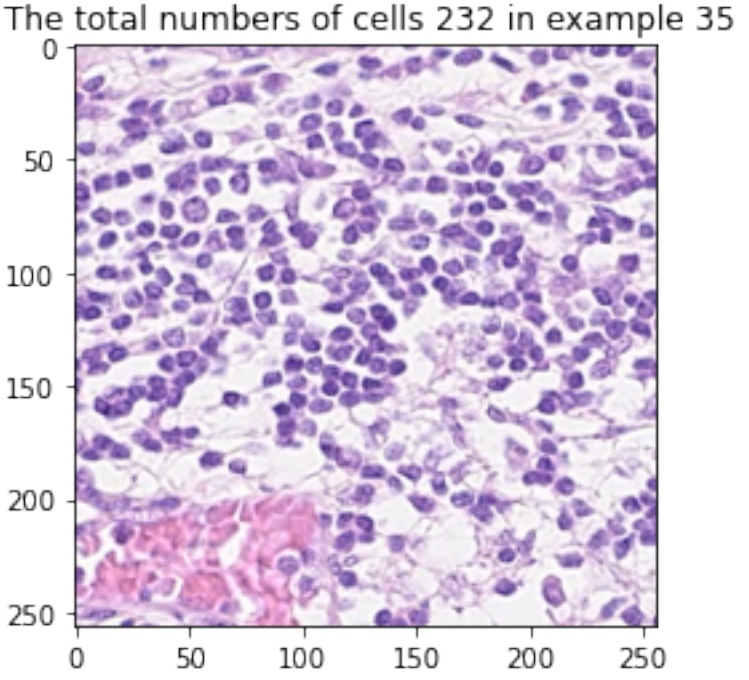

```
Typeepithelial    26
lymphocyte     2
plasma         8
connective     4
Name: 1896, dtype: int64
```



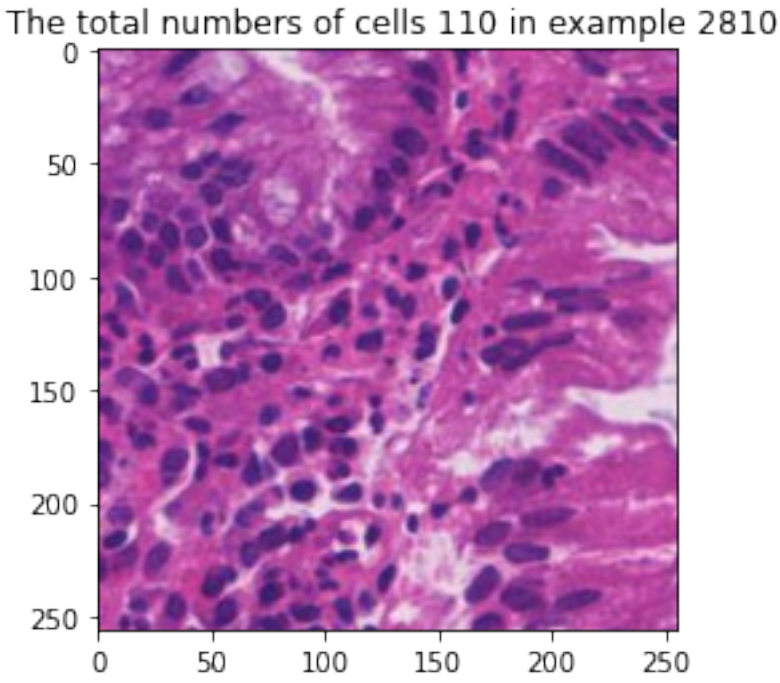

```
Typeepithelial    53
lymphocyte    16
plasma         7
connective    27
Name: 511, dtype: int64
```



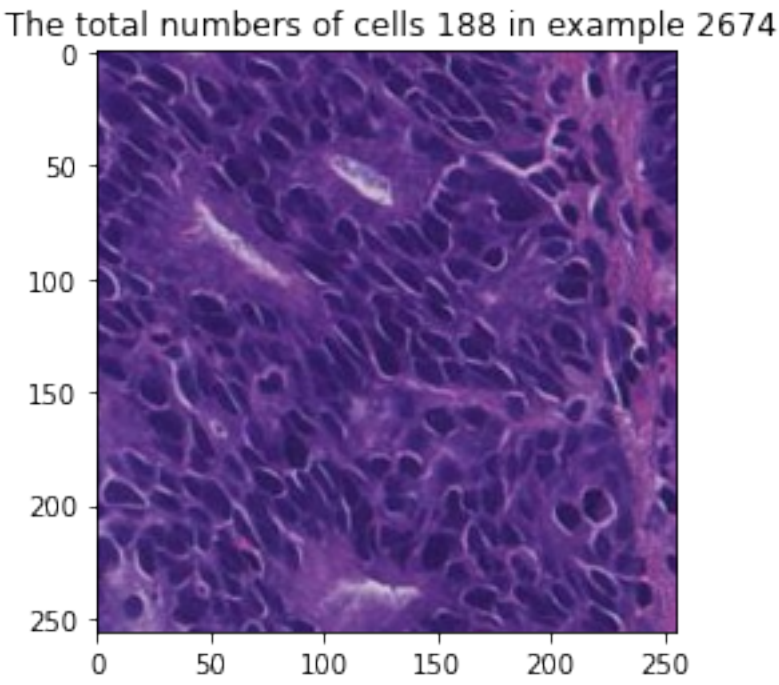

```
Typelymphocyte    161
plasma              3
connective         68
Name: 35, dtype: int64
```



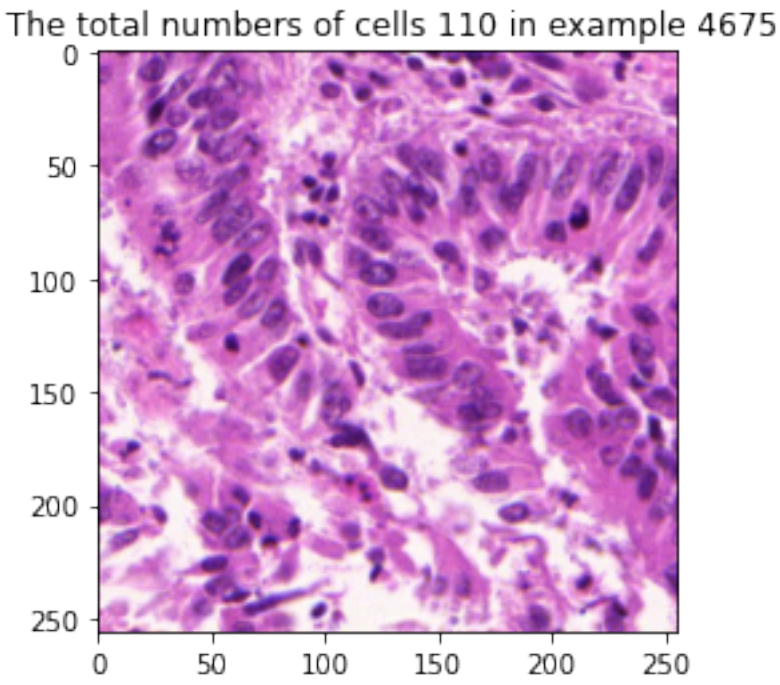
```
Typeneutrophil     4
epithelial    43
lymphocyte    41
plasma        10
connective    12
Name: 2810, dtype: int64
```



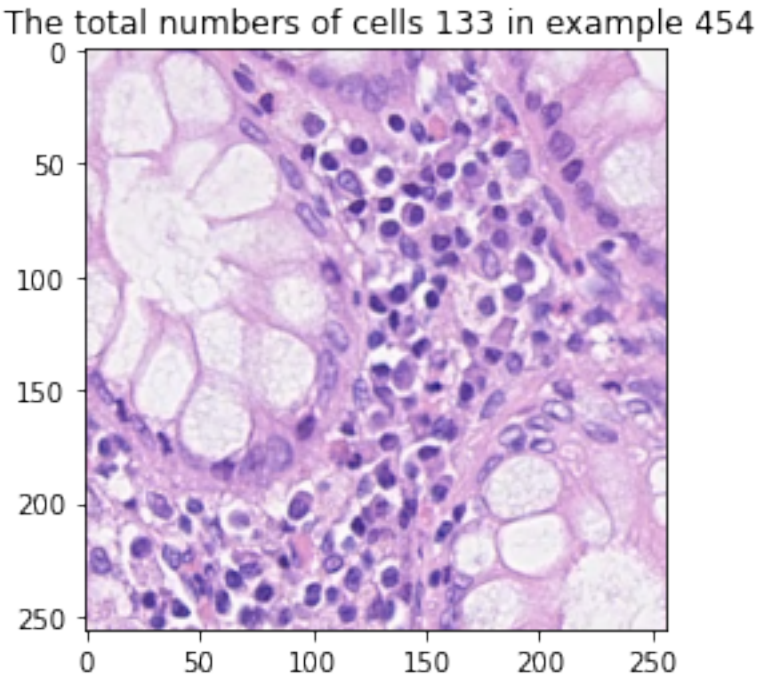

```
Typeepithelial    168
lymphocyte     13
plasma          1
connective      6
Name: 2674, dtype: int64
```



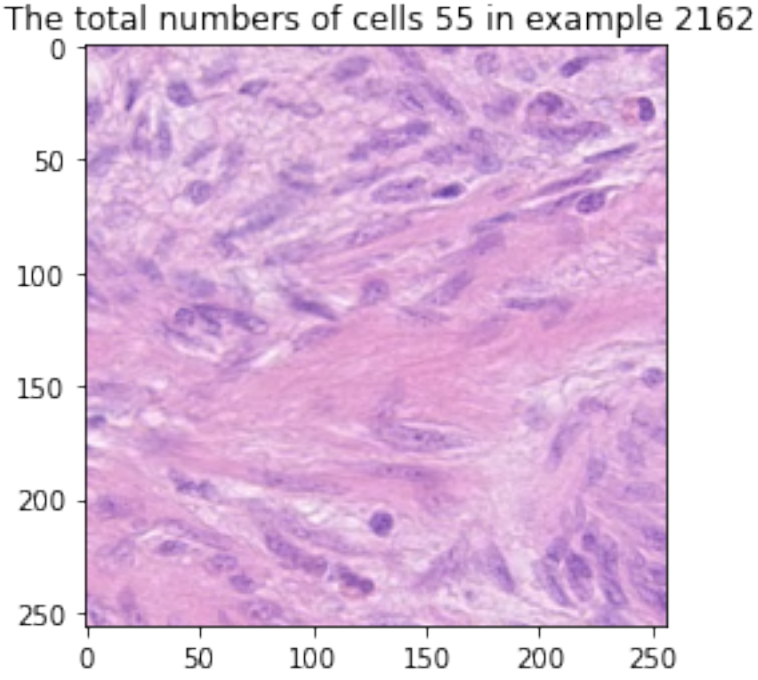

```
Typeneutrophil     1
epithelial    69
lymphocyte    28
plasma         6
eosinophil     1
connective     5
Name: 4675, dtype: int64
```



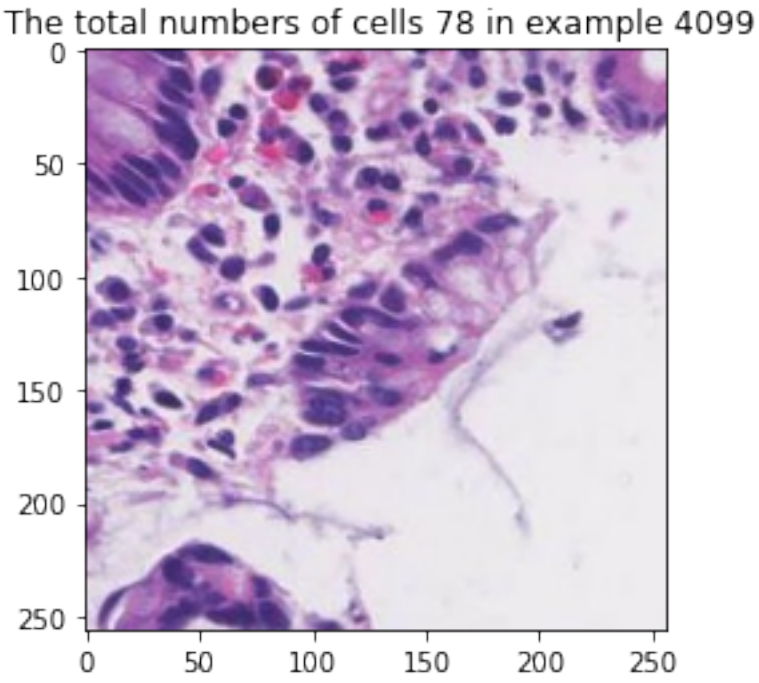

```
Typeepithelial    33
lymphocyte    51
plasma        23
connective    26
Name: 454, dtype: int64
```



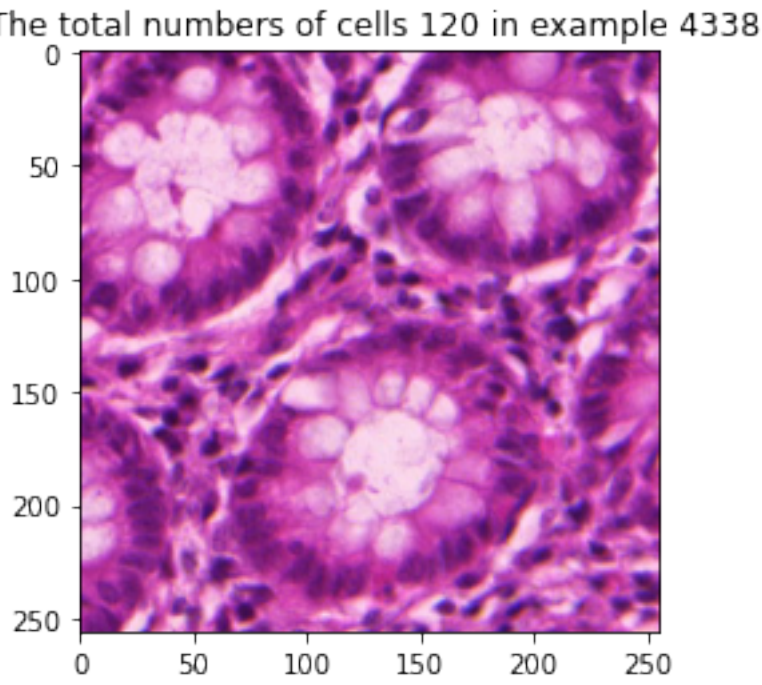

```
Typelymphocyte     2
eosinophil     2
connective    51
Name: 2162, dtype: int64
```



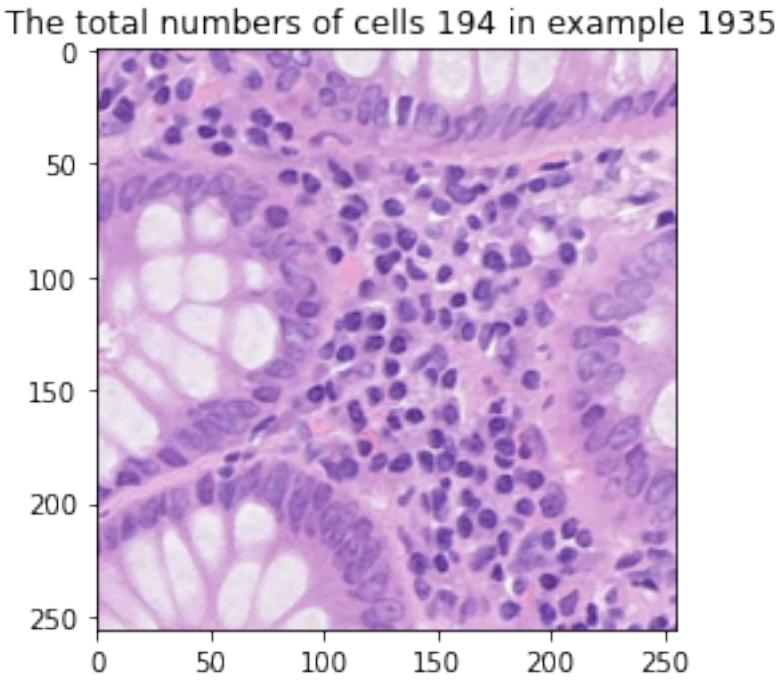
```
Typeepithelial    27
lymphocyte    30
plasma         9
eosinophil     3
connective     9
Name: 4099, dtype: int64
```



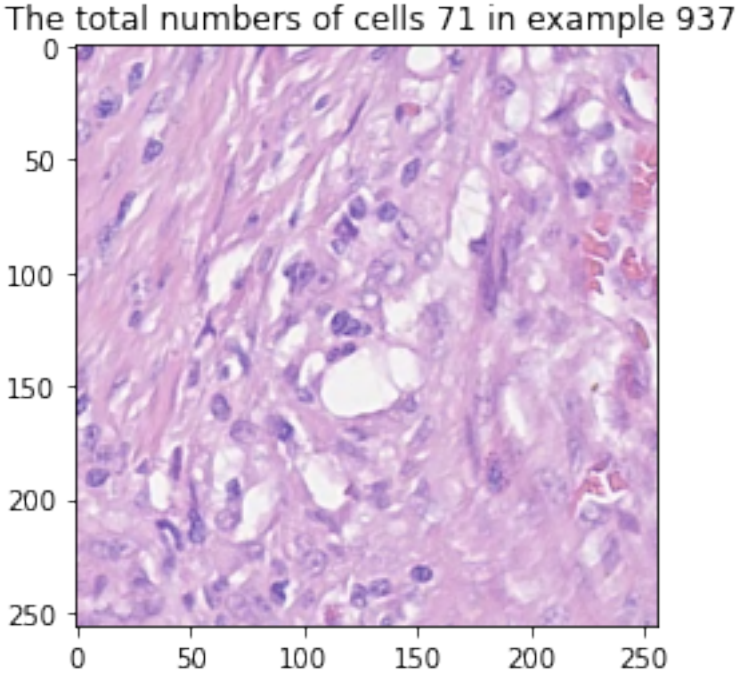

```
Typeepithelial    69
lymphocyte    19
plasma         8
connective    24
Name: 4338, dtype: int64
```



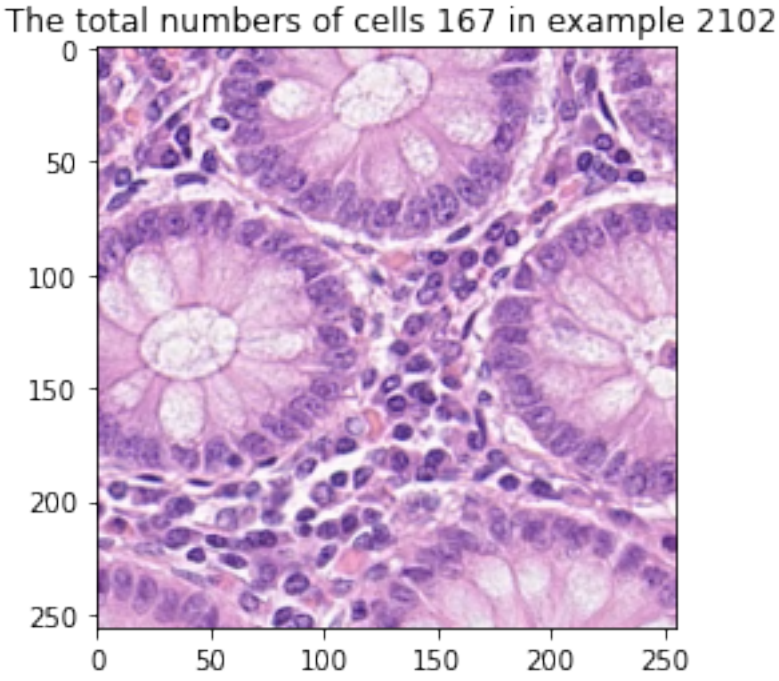

```
Typeepithelial    75
lymphocyte        74
plasma            17
eosinophil         1
connective        27
Name: 1935, dtype: int64
```



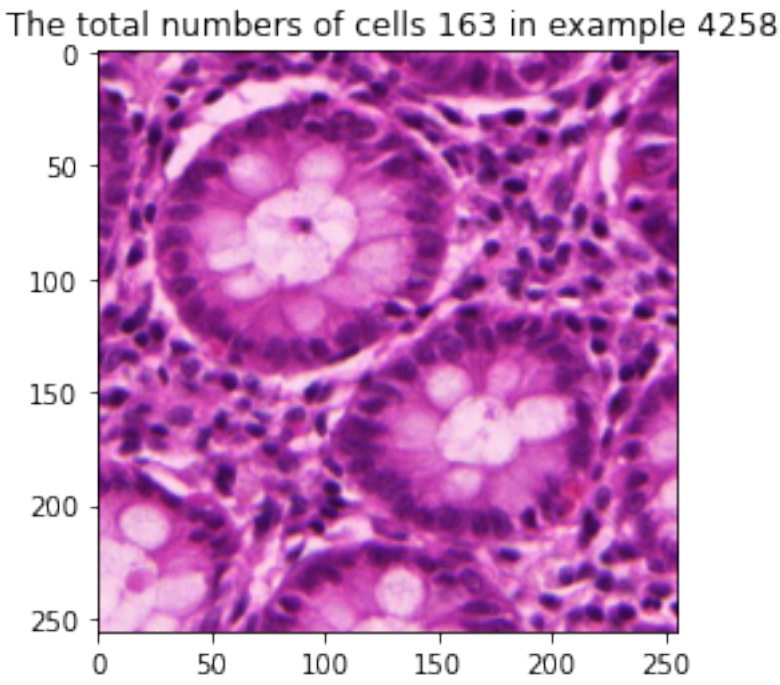

```
Typelymphocyte    12
eosinophil     1
connective    58
Name: 937, dtype: int64
```



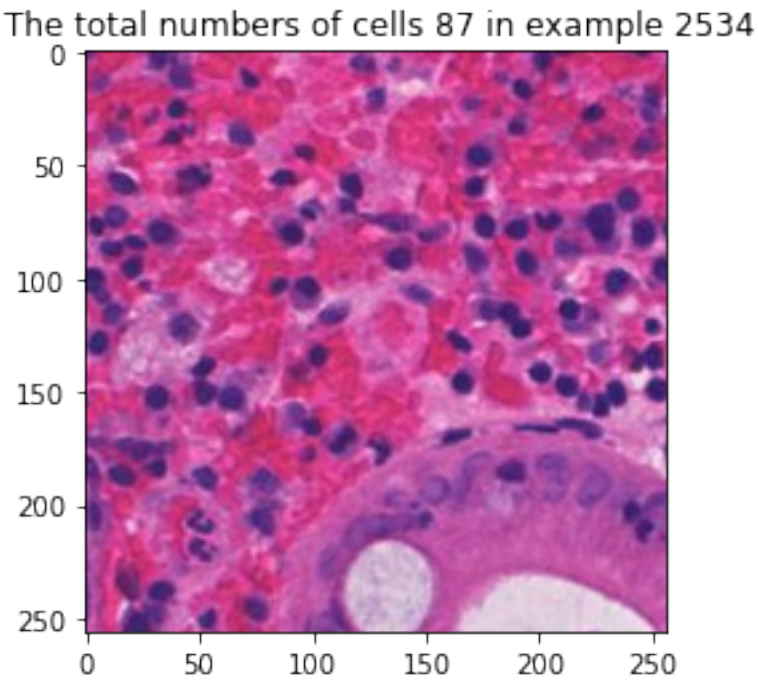
```
Typeepithelial    77
lymphocyte    30
plasma        34
eosinophil     1
connective    25
Name: 2102, dtype: int64
```



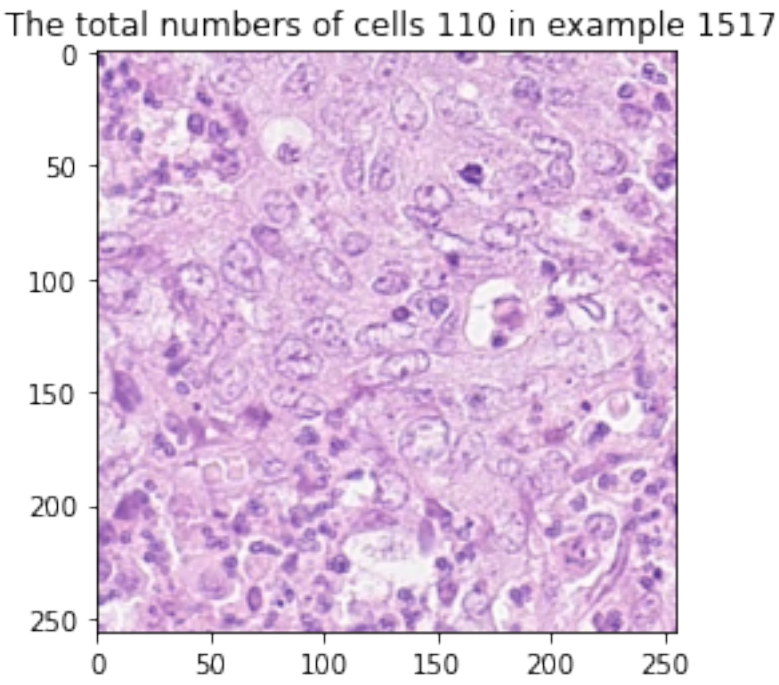

```
Typeepithelial    70
lymphocyte    51
plasma        14
connective    28
Name: 4258, dtype: int64
```



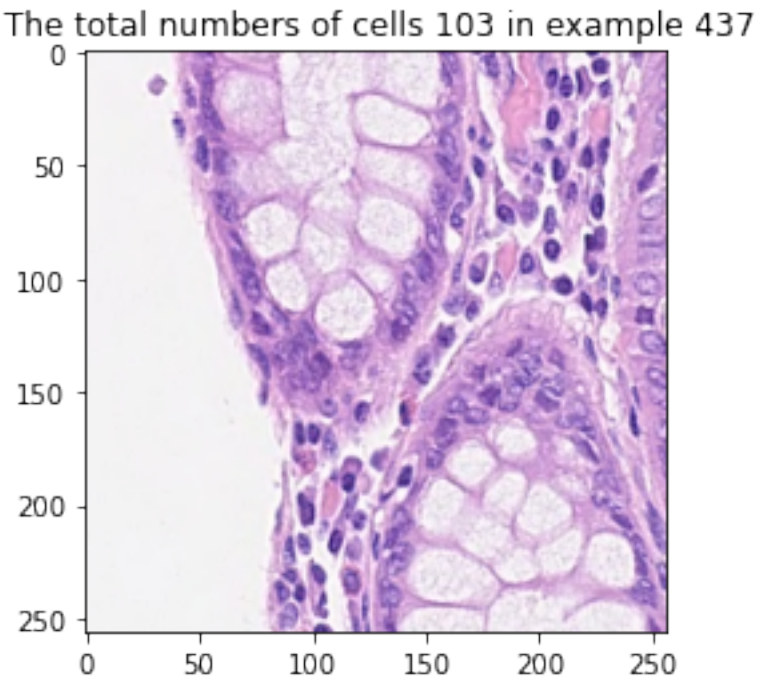

```
Typeneutrophil     2
epithelial         9
lymphocyte        45
plasma            14
connective        17
Name: 2534, dtype: int64
```



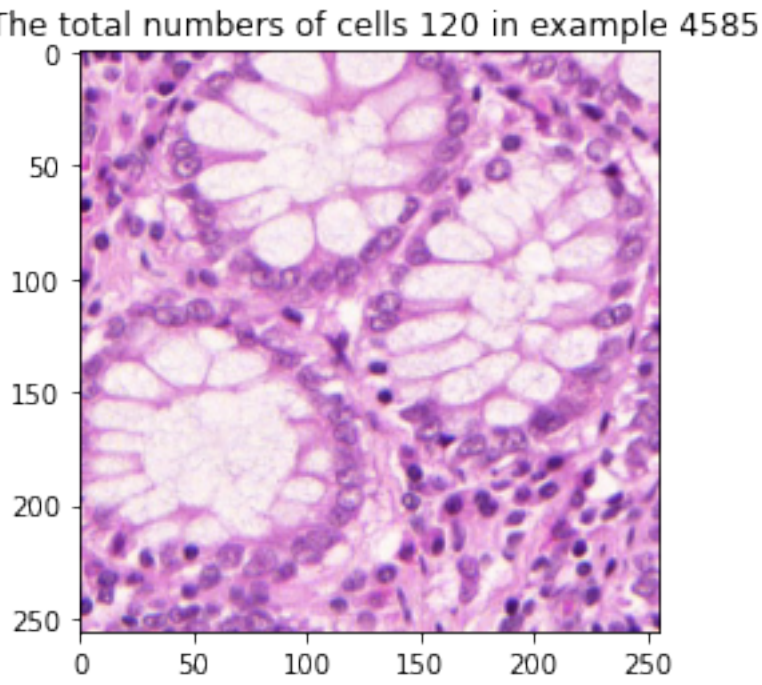

```
Typeneutrophil    12
epithelial        56
lymphocyte        26
plasma             1
connective        15
Name: 1517, dtype: int64
```



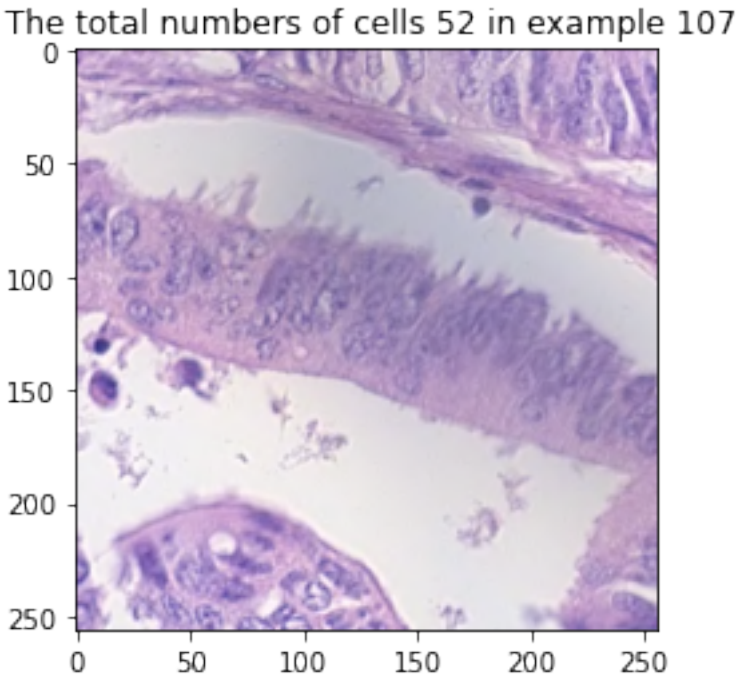

```
Typeepithelial    49
lymphocyte    18
plasma        16
eosinophil     1
connective    19
Name: 437, dtype: int64
```




```
Typeepithelial    59
lymphocyte        23
plasma            16
eosinophil         1
connective        21
Name: 4585, dtype: int64
```




```
Typeepithelial    48
lymphocyte     1
plasma         1
connective     2
Name: 107, dtype: int64
```



1.0.3 iii. For each fold, plot the histogram of counts of each cell type separately as well as the total number of cells (7 plots in total). How many images have counts within each of the following bins?

Note: I also include the counts which the total cell number exceed 100. The code can be divided into several sections.

First, the script defines the names of the cell types and the three folds used for counting and visualization.

Next, the number of cells falling within each cell count range is counted using predefined bins, and the results are stored in a DataFrame object.

Subsequently, histograms are plotted using the plt.hist() function for each cell type in each fold as well as the total number of cells, and printed out.

Finally, the code outputs a plot displaying the number of cells in each cell count range and for each cell type in each of the three folds.

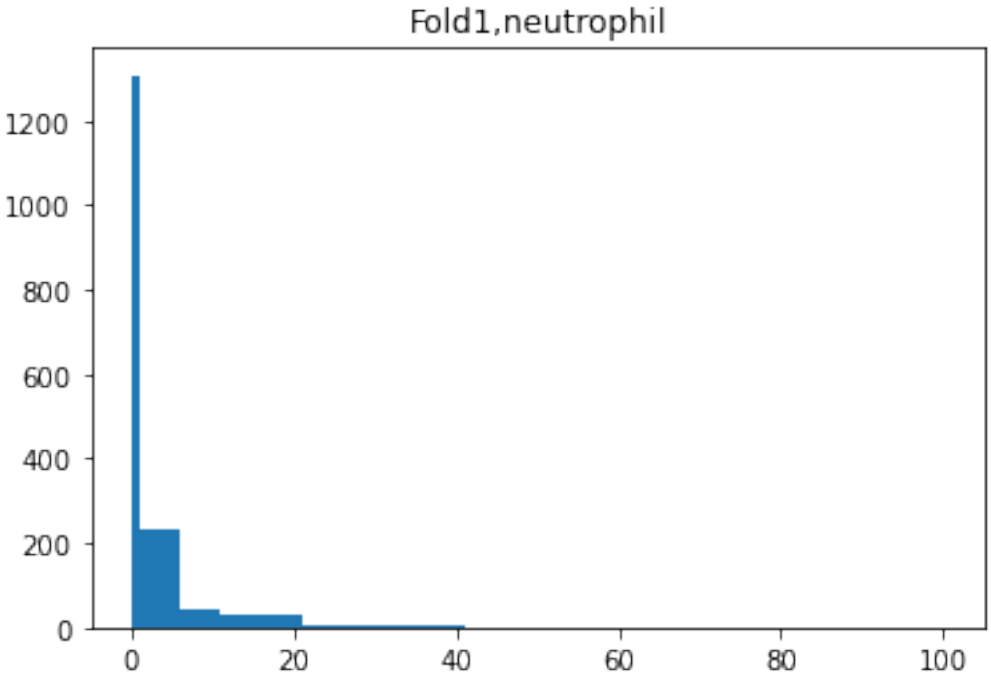
Through these statistics and visualizations, we can get a general idea of the distribution of different cell types and the total number of cells in the training set. Specifically, we can observe that some cell types are less abundant while others are more abundant, and we can also see that the distribution of cell numbers varies slightly across different folds. This information can be useful for subsequent model training and evaluation.

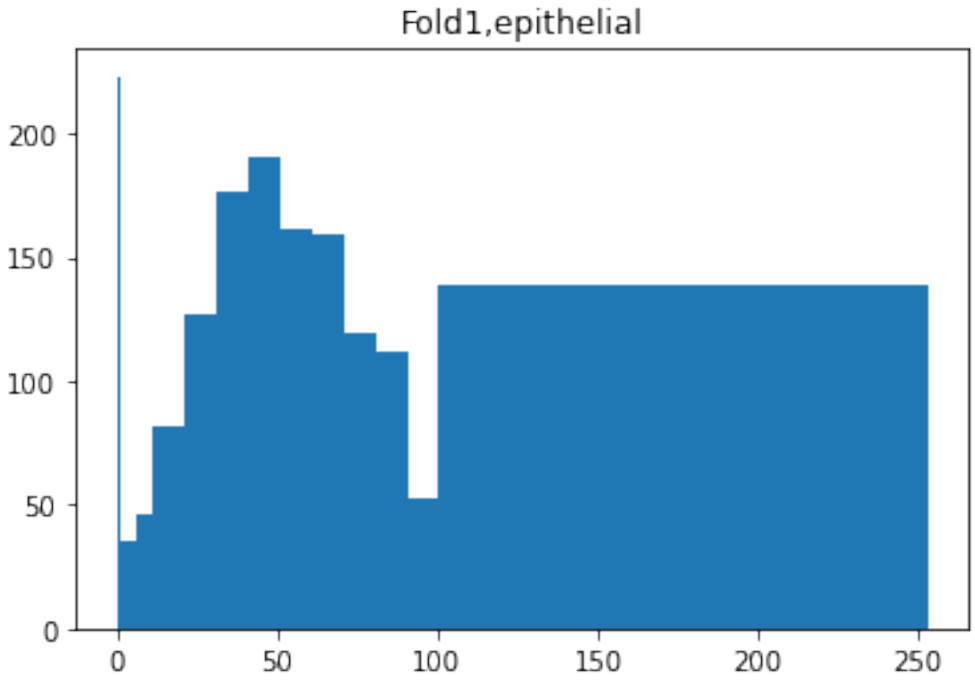
[129]:
```
type_name =␣
↪["neutrophil","epithelial","lymphocyte","plasma","eosinophil","connective"]
```

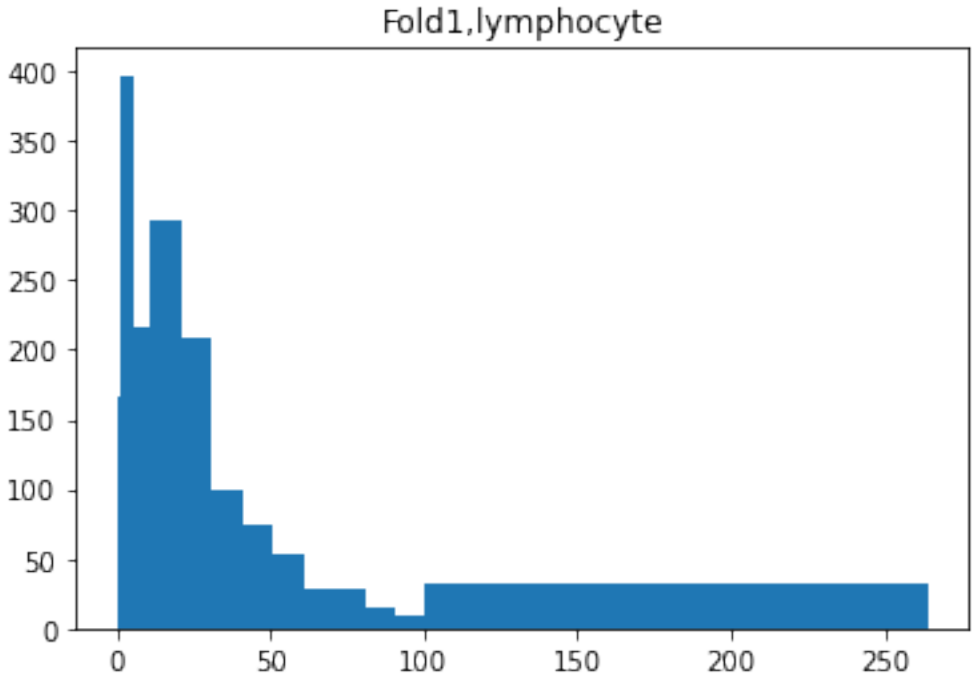
```
fold = [1,2,3]
# Print the counts in each bin
bins = [0,1,6, 11, 21, 31, 41, 51, 61, 71, 81, 91, 101, float('inf')]
fold1_bin=pd.value_counts(pd.cut(Y[F==1].apply(lambda x:x.sum(),axis␣
 ↪=1),bins,right=False),sort=False).values

fold2_bin=pd.value_counts(pd.cut(Y[F==2].apply(lambda x:x.sum(),axis␣
 ↪=1),bins,right=False),sort=False).values

fold3_bin=pd.value_counts(pd.cut(Y[F==3].apply(lambda x:x.sum(),axis␣
 ↪=1),bins,right=False),sort=False).values

bin=pd.value_counts(pd.cut(Y[F==1].apply(lambda x:x.sum(),axis␣
 ↪=1),bins,right=False),sort=False).index.tolist()

fold_bin=[fold1_bin,fold2_bin,fold3_bin]
df=DataFrame(fold_bin,index=['fold 1','fold 2','fold 3'],columns=bin).T
print(df)


Y_fold1=Y[F==1]
To_Y_fold1=Y_fold1.apply(lambda x:x.sum(),axis =1)


for i in fold:
    for j in range(len(type_name)):
        a =  Y[F==i][type_name[j]]
        n, bins_1, patches=plt.hist(a,bins=[0,1,6, 11, 21, 31, 41, 51, 61, 71,␣
 ↪81, 91, 100, np.max(np.concatenate([a,[100]]))])
        plt.title("Fold"+str(i)+","+str(type_name[j]))
        plt.show()
    n, bins_2, patches=plt.hist(To_Y_fold1,bins=[0,1,6, 11, 21, 31, 41, 51, 61,␣
 ↪71, 81, 91, 100, np.max(np.concatenate([To_Y_fold1,[100]]))])
    plt.title("Overall in Fold"+str(i))
    plt.show()
```

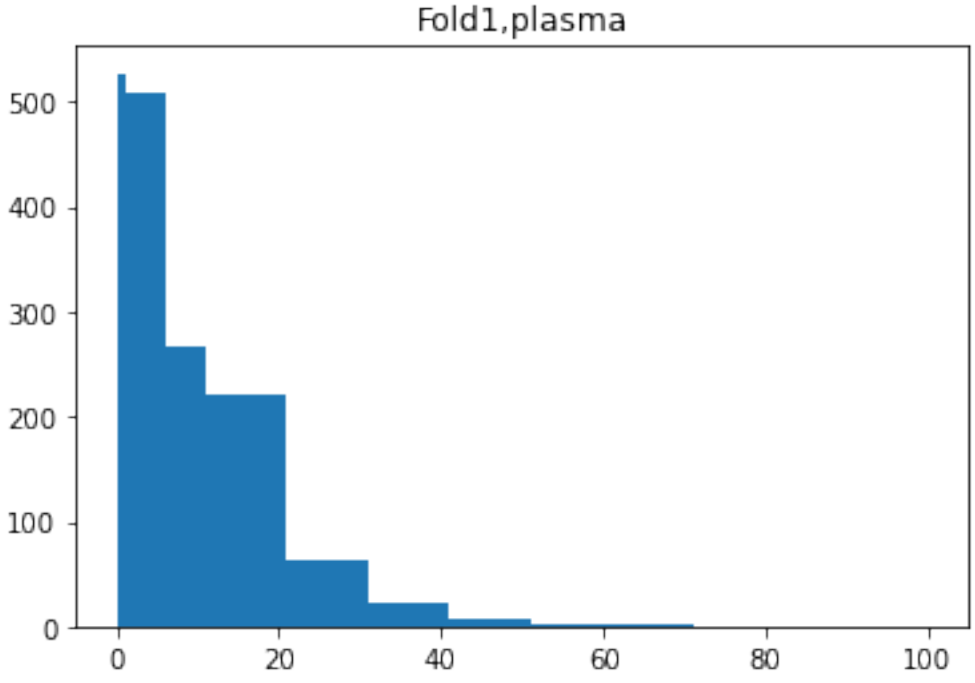
	fold 1	fold 2	fold 3
[0.0, 1.0)	39	54	57
[1.0, 6.0)	17	13	31
[6.0, 11.0)	15	30	36
[11.0, 21.0)	23	51	85
[21.0, 31.0)	47	47	90
[31.0, 41.0)	43	66	97
[41.0, 51.0)	65	107	71
[51.0, 61.0)	83	137	105
[61.0, 71.0)	94	134	118
[71.0, 81.0)	132	135	124

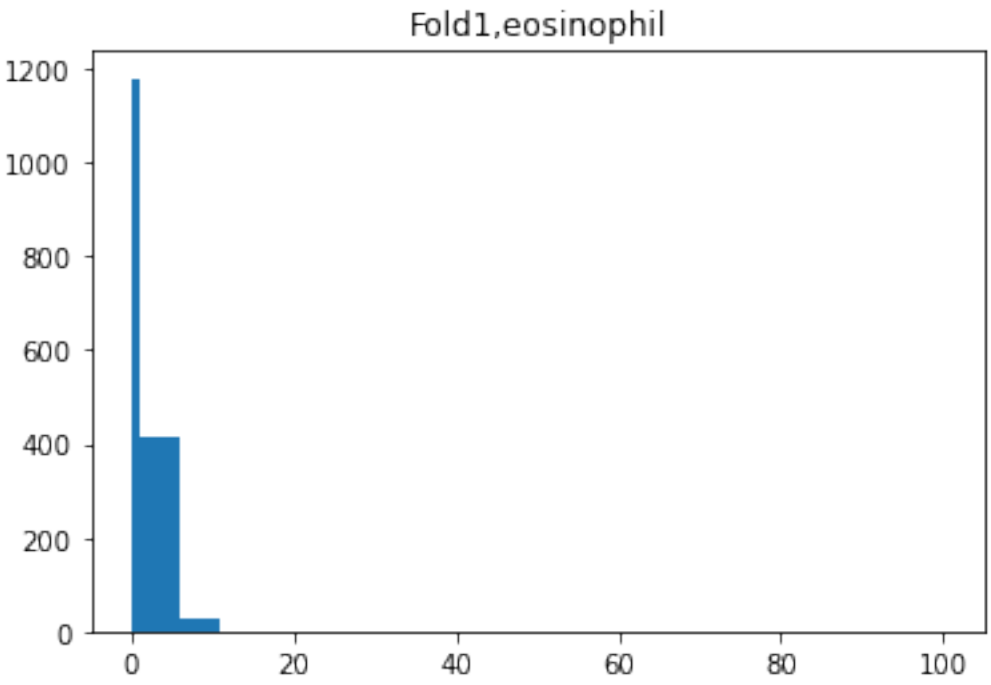
[81.0, 91.0)	164	162	133
[91.0, 101.0)	132	160	128
[101.0, inf)	768	655	533

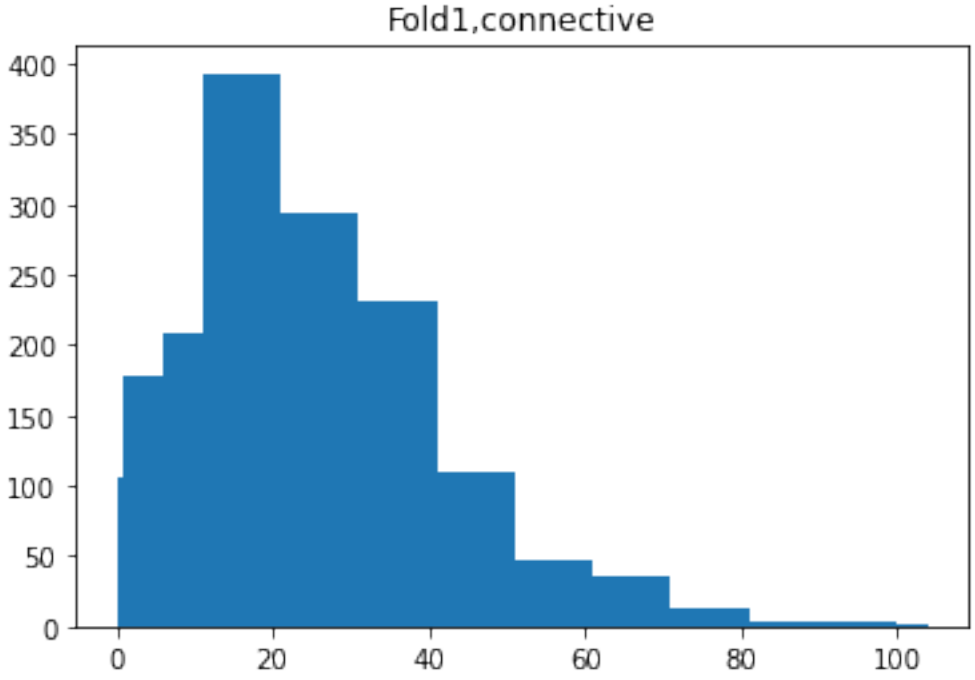


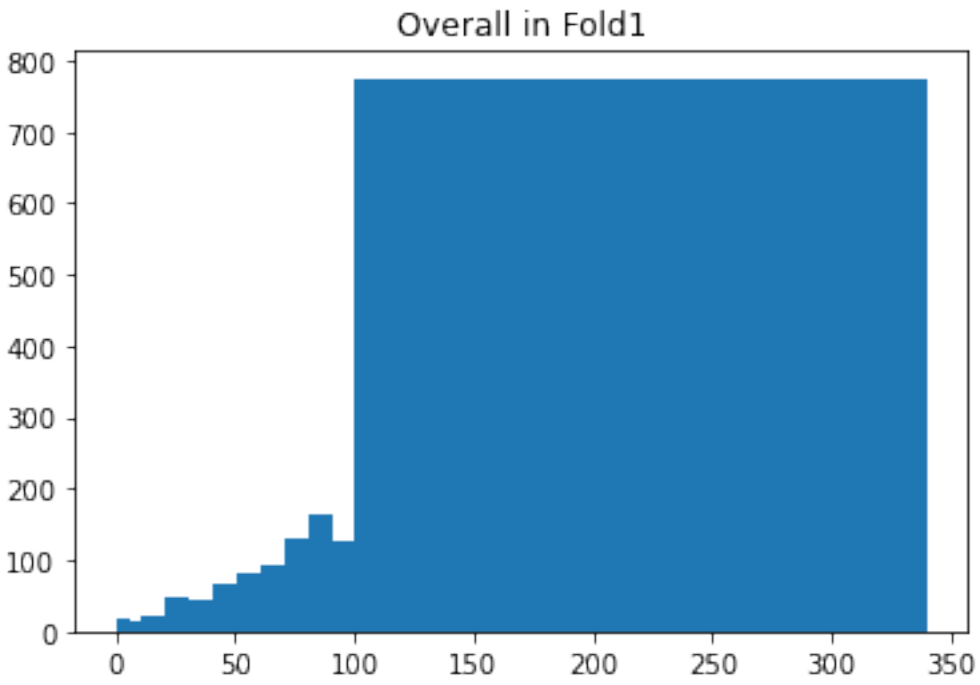


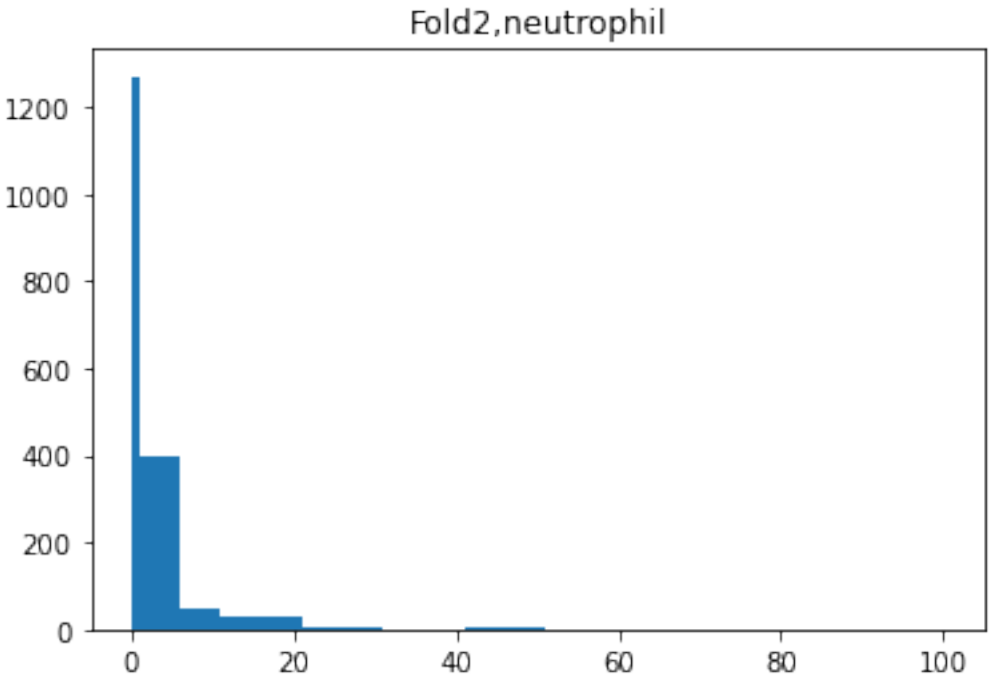

Fold1,lymphocyte
400
350
300
250
200
150
100
50
0
0
50
100
150
200
250

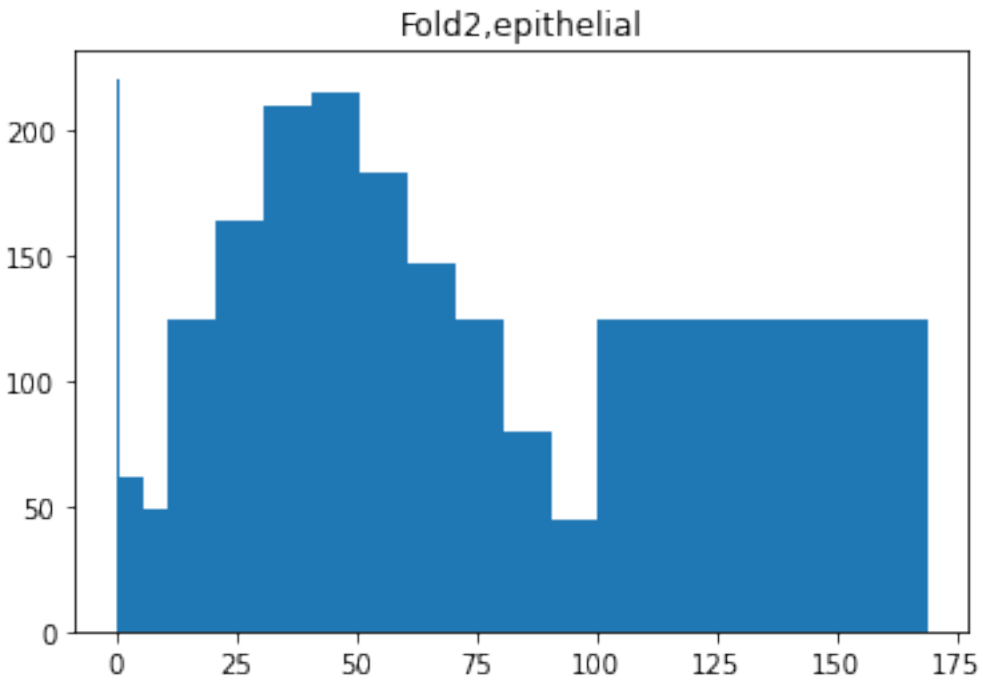

Fold1,plasma
500
400
300
200
100
0
0
20
40
60
80
100

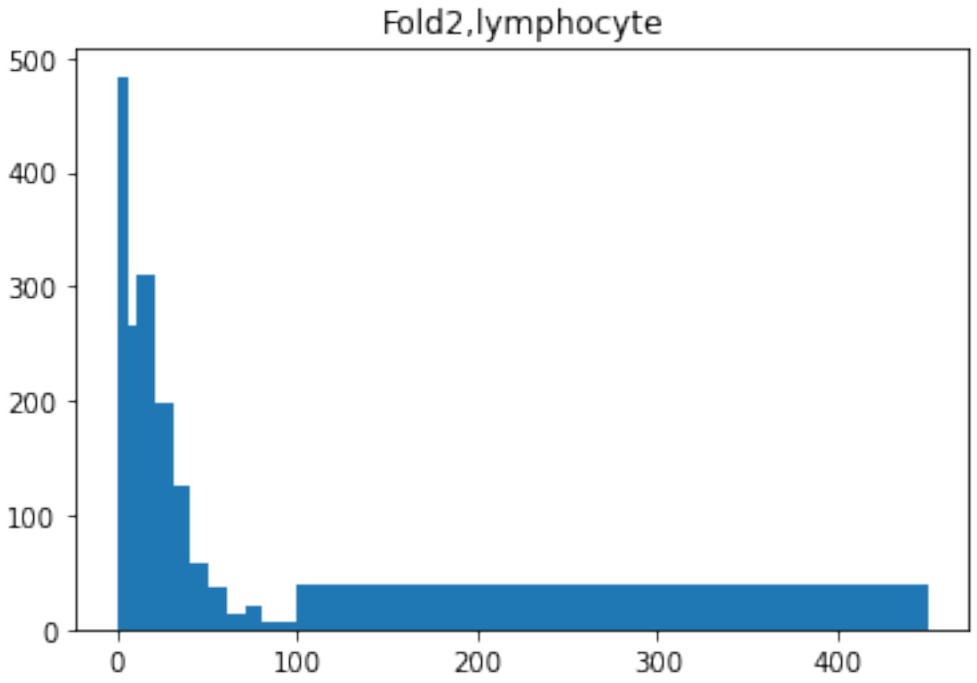

Fold1,eosinophil
1200
1000
800
600
400
200
0
0
20
40
60
80
100

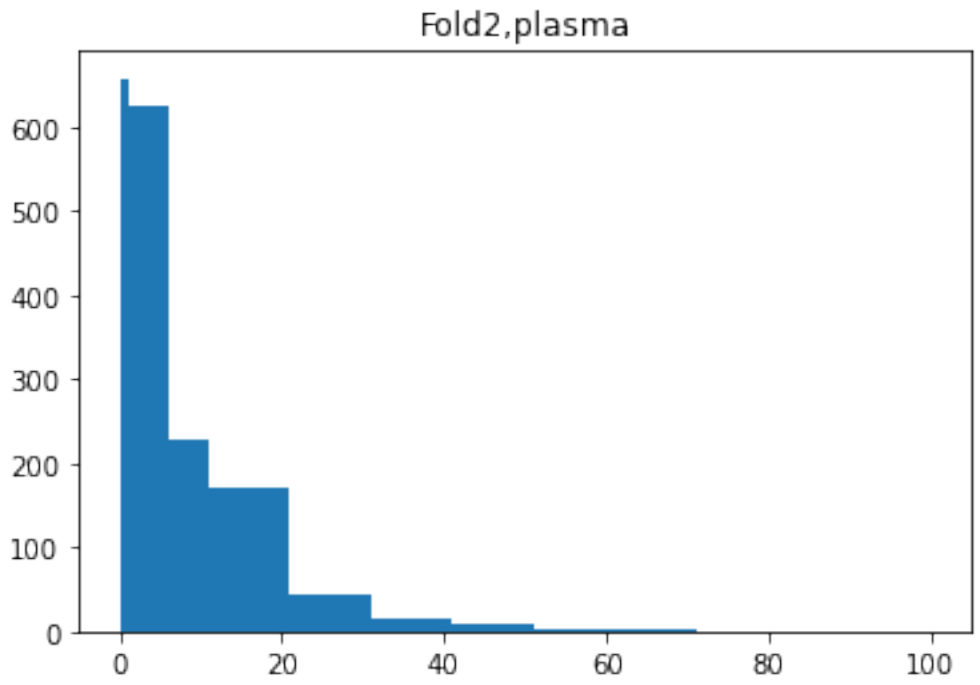

Fold1,connective
400
350
300
250
200
150
100
50
0
0
20
40
60
80
100

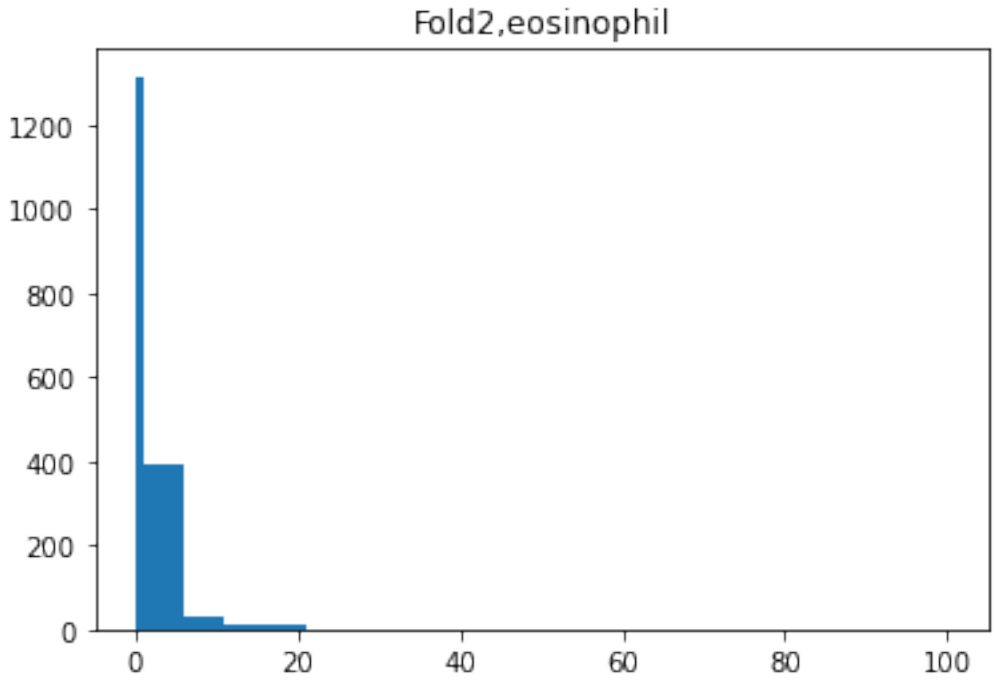

Overall in Fold1
800
700
600
500
400
300
200
100
0
0
50
100
150
200
250
300
350

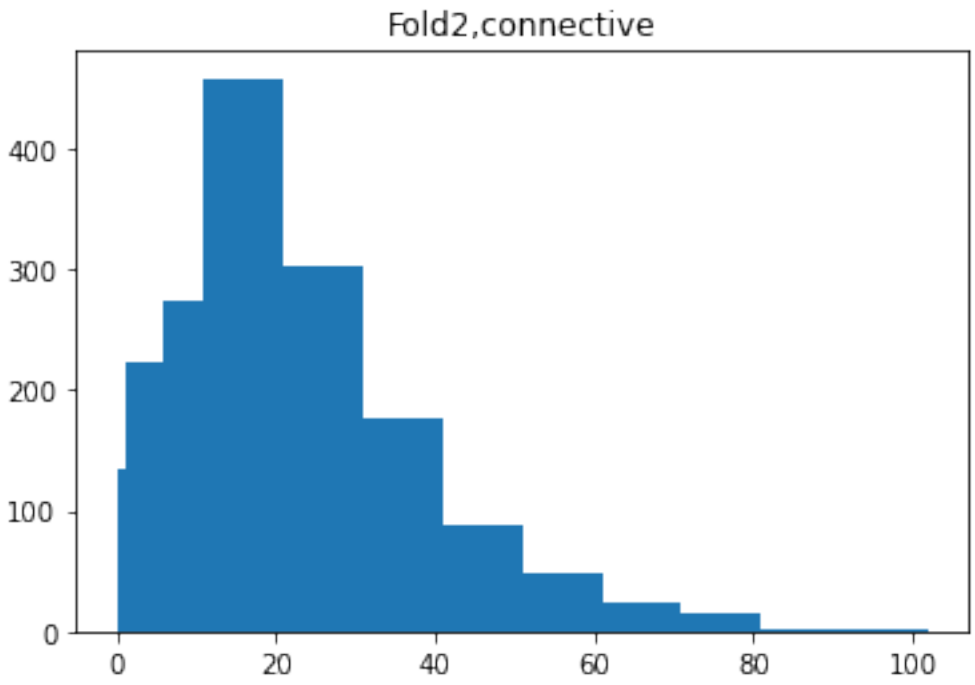

Fold2,neutrophil
1200
1000
800
600
400
200
0
0
20
40
60
80
100

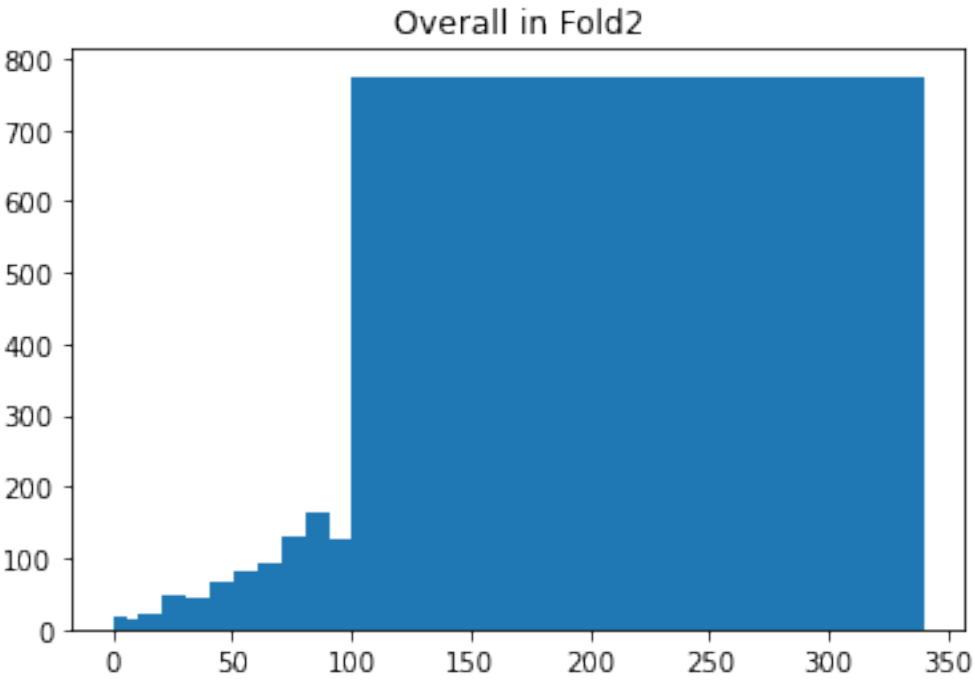

Fold2,epithelial
200
150
100
50
0
0
25
50
75
100
125
150
175

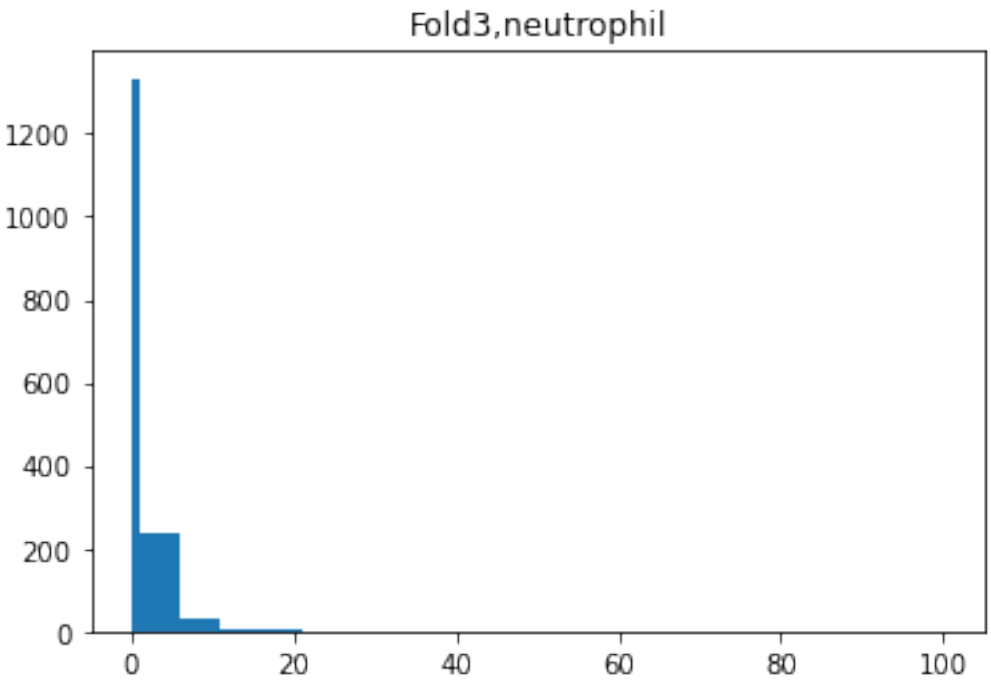

Fold2,lymphocyte
500
400
300
200
100
0
0
100
200
300
400

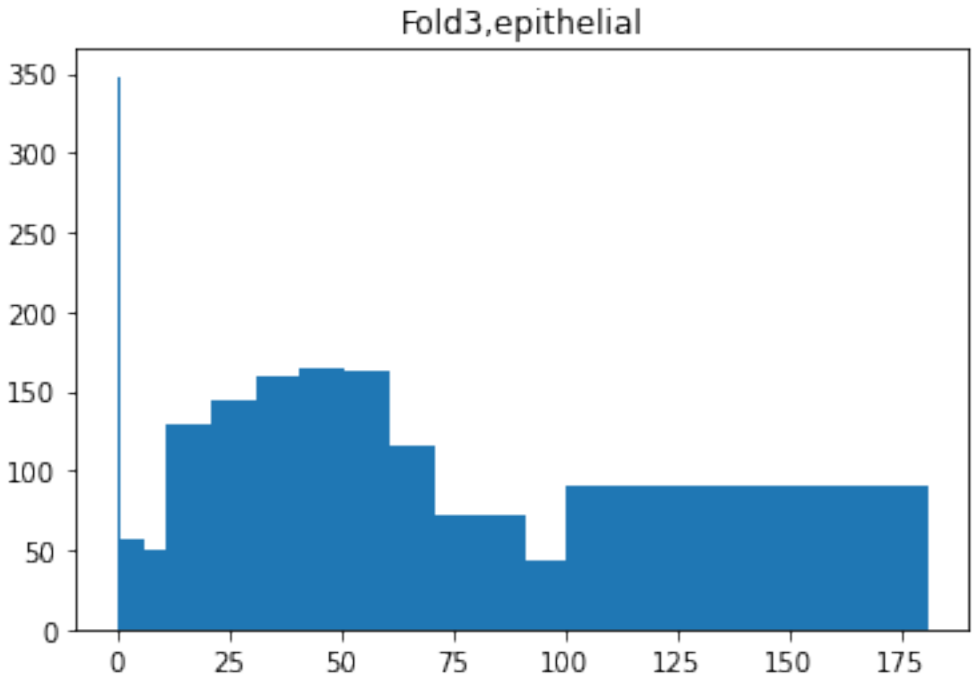

Fold2,plasma
600
500
400
300
200
100
0
0
20
40
60
80
100

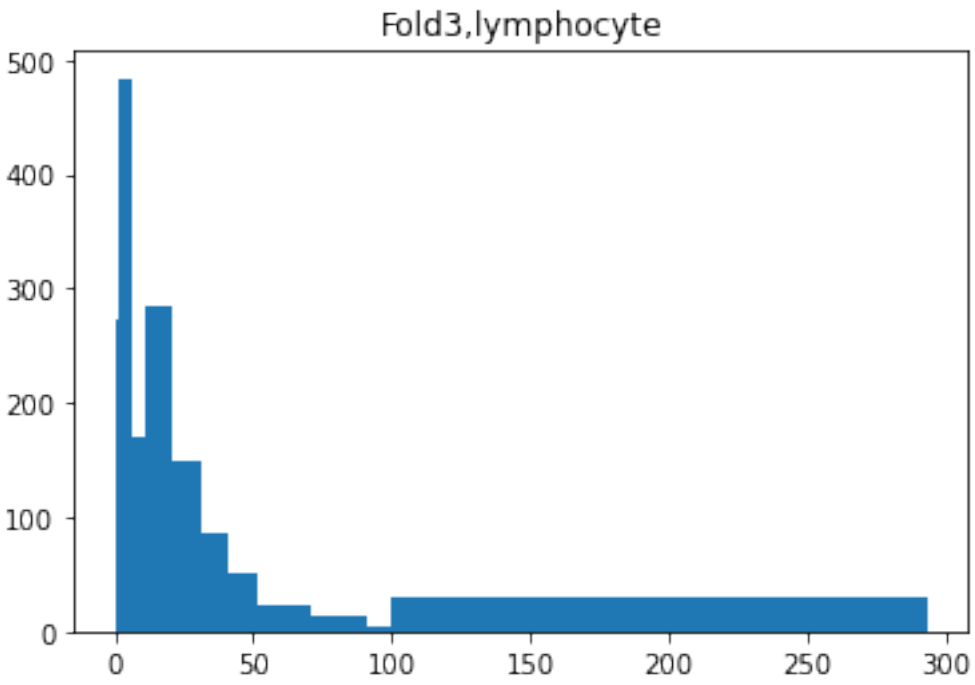

Fold2,eosinophil
1200
1000
800
600
400
200
0
0
20
40
60
80
100

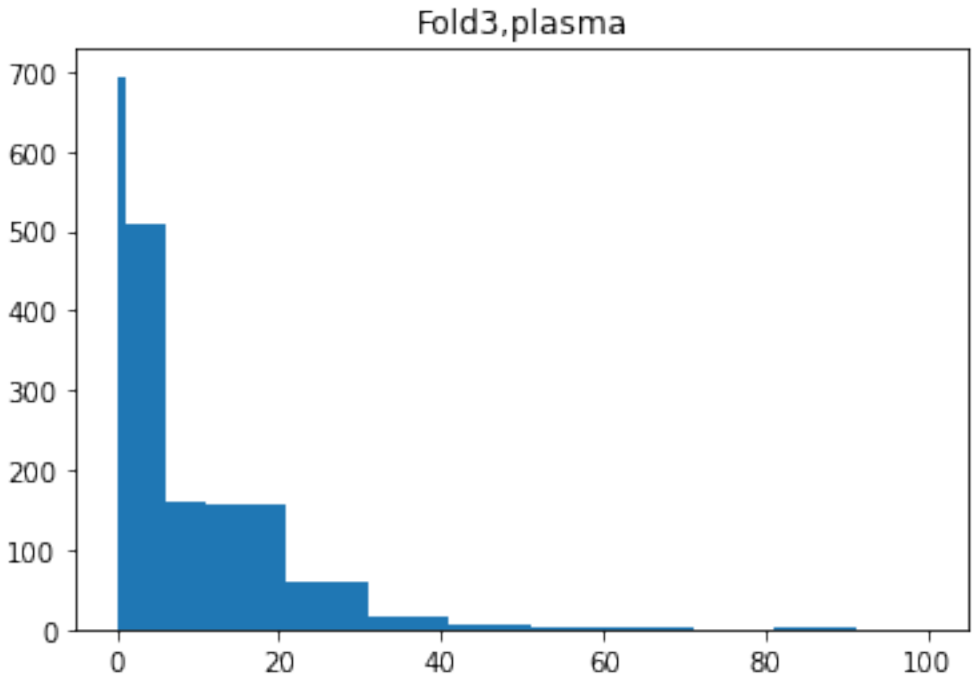

Fold2,connective
400
300
200
100
0
0
20
40
60
80
100

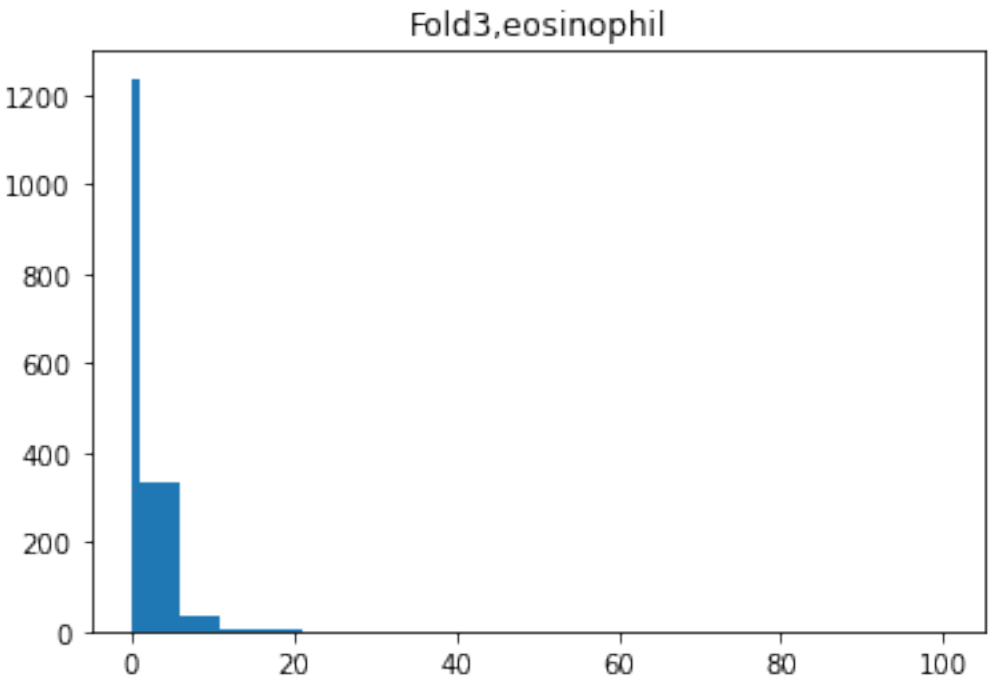

Overall in Fold2
800
700
600
500
400
300
200
100
0
0
50
100
150
200
250
300
350

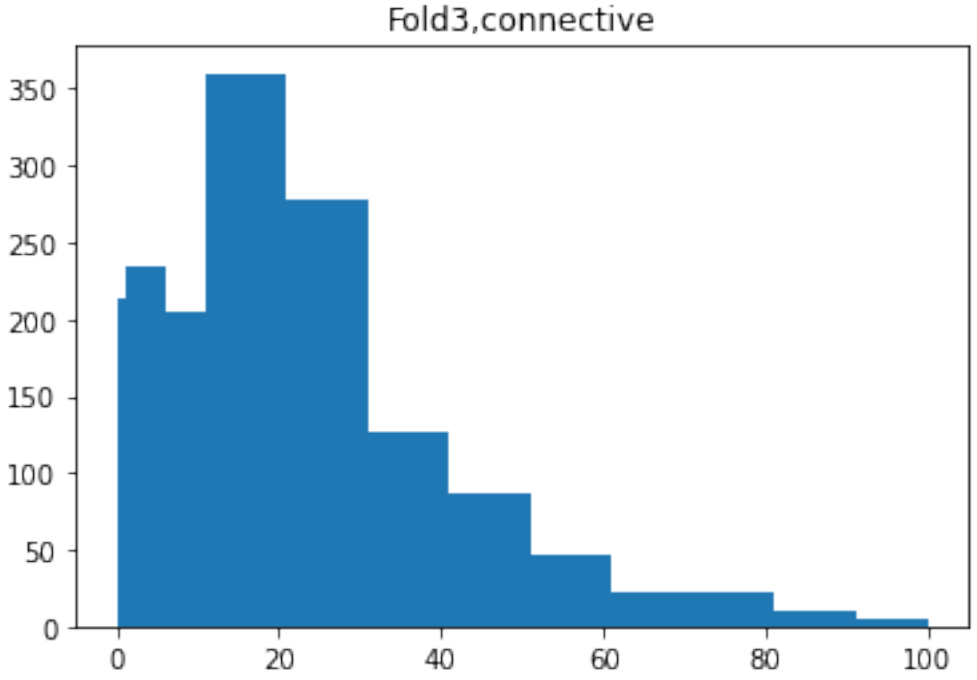

Fold3,neutrophil
1200
1000
800
600
400
200
0
0
20
40
60
80
100

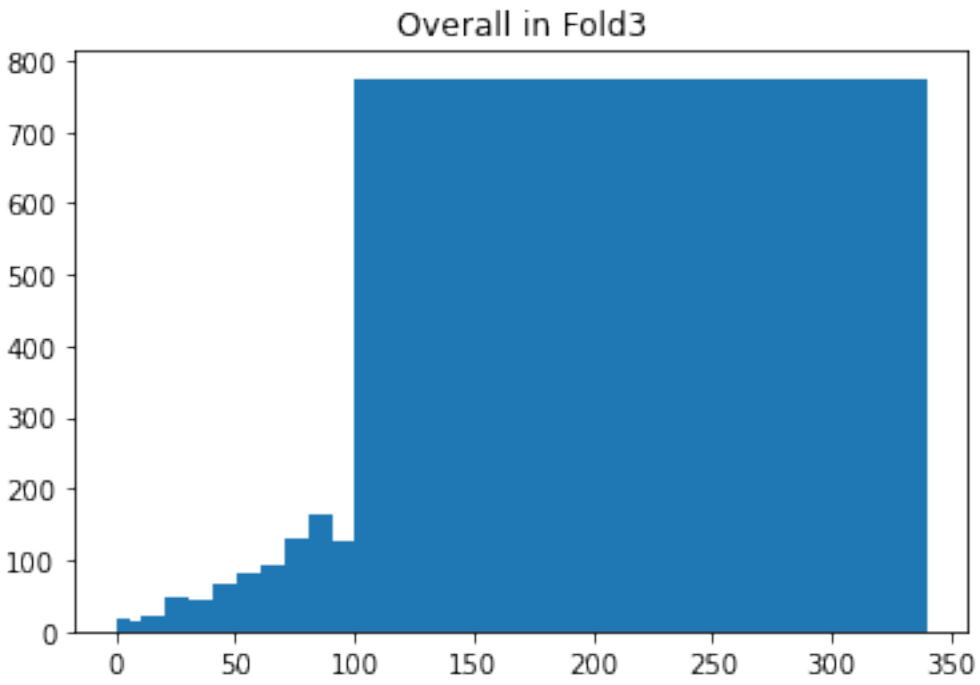

Fold3,epithelial
350
300
250
200
150
100
50
0
0
25
50
75
100
125
150
175


Fold3,lymphocyte
500
400
300
200
100
0
0
50
100
150
200
250
300


Fold3,plasma
700
600
500
400
300
200
100
0
0
20
40
60
80
100


Fold3,eosinophil
1200
1000
800
600
400
200
0
0
20
40
60
80
100


Fold3,connective
350
300
250
200
150
100
50
0
0
20
40
60
80
100

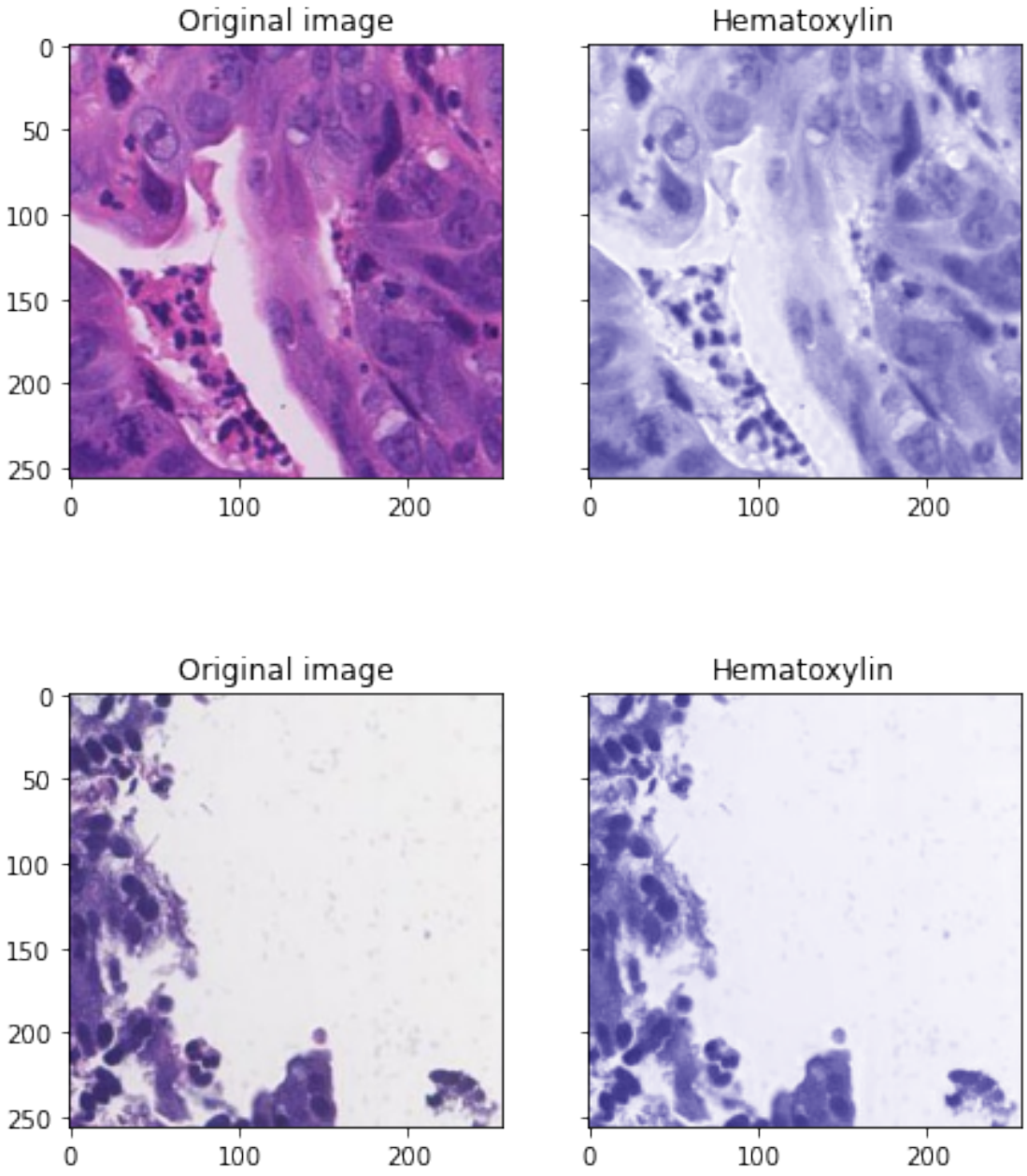


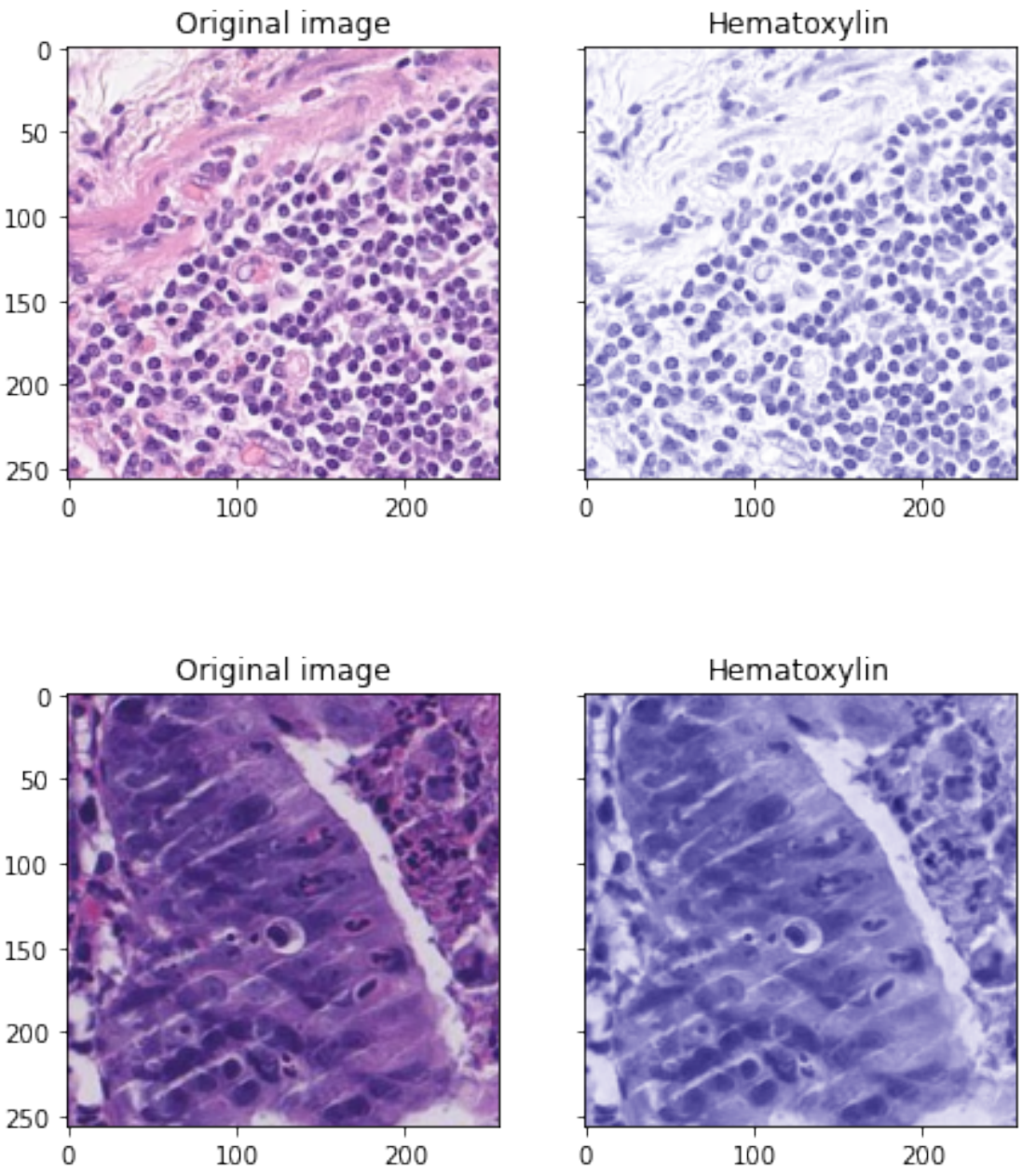
1.0.4 iv. Pre-processing: Convert and show a few images from RGB space to HED space and show the H-channel which should indicate cellular nuclei. For this purpose, you can use the color separation notebook available here: https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_ihc_color_separation.html

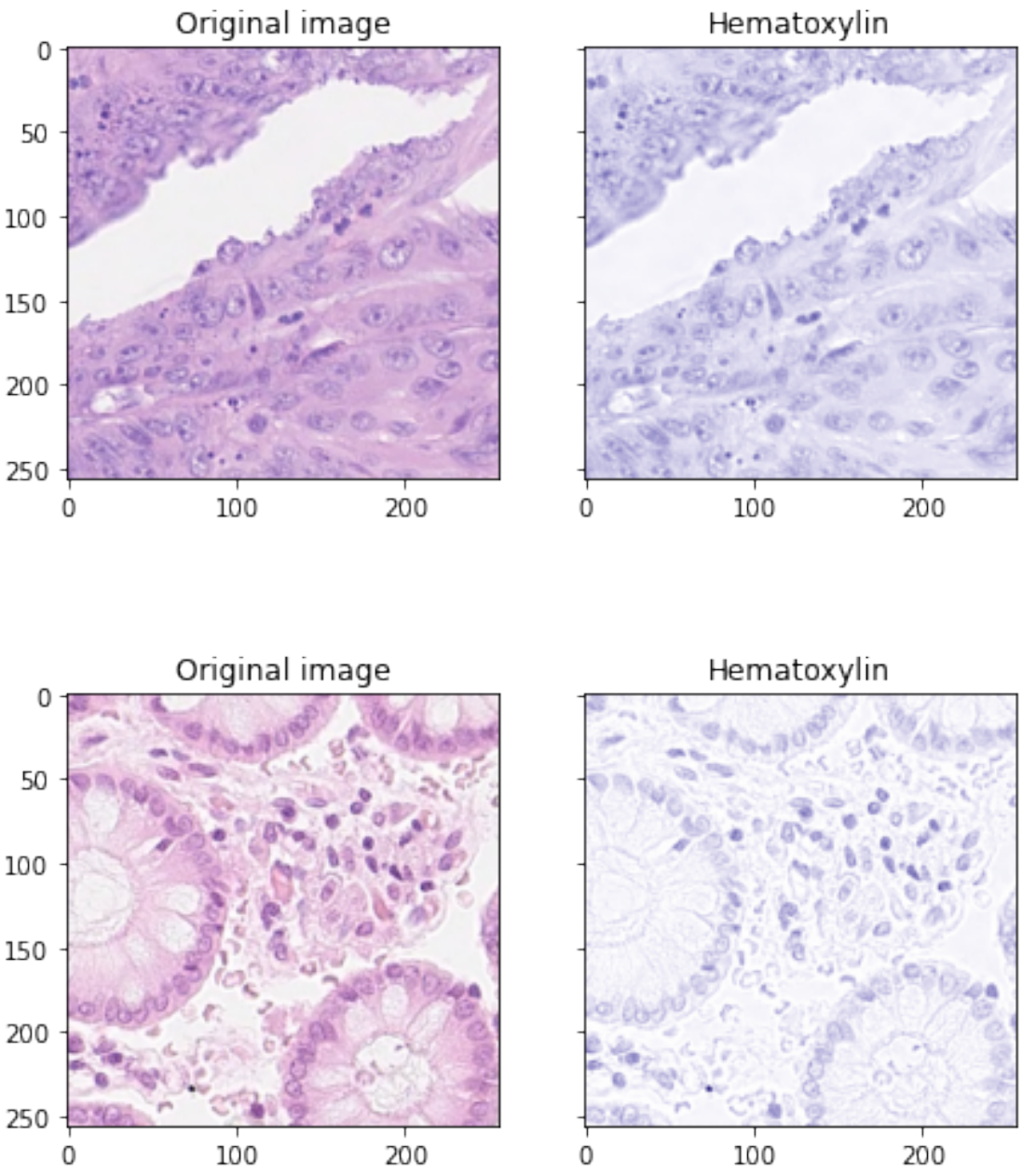
What this code does is to select 10 samples at random from the image dataset, convert them to the HED (Hematoxylin-Eosin-DAB) colour space and display the original image and Hematoxylin channel for one of the samples. By looping, for each selected sample, the code calculates the HED colour space and converts the result to an RGB image, finally displaying the original image and the Hematoxylin image in a chart containing two subplots. This can be used to observe the structure and position of the cell nuclei in the image.

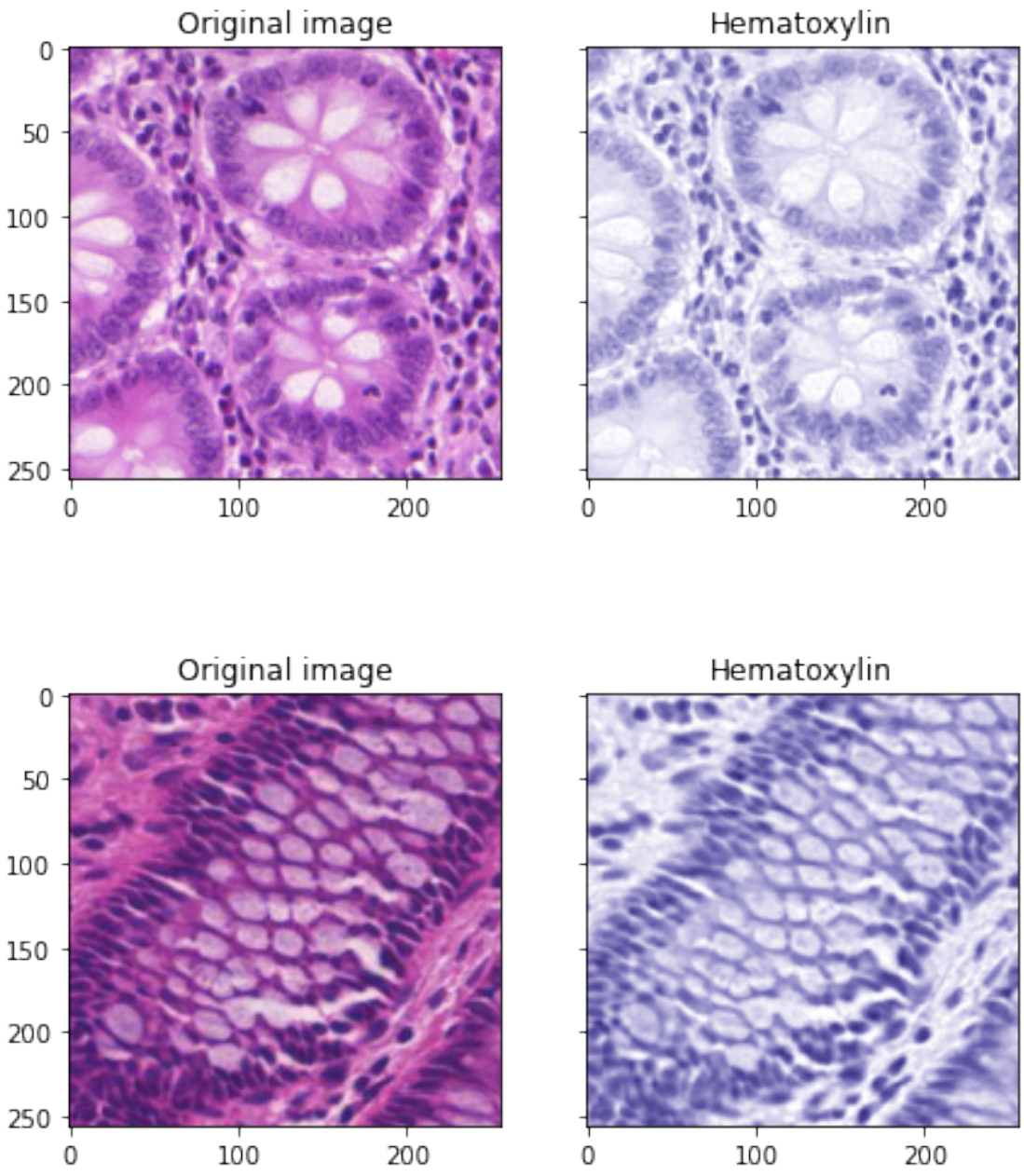
[9]:
```
idx = np.random.randint(X.shape[0], size=10)
idx

ihc_hed = []
for i in range(len(X)):
    ihc_hed.append(rgb2hed(X[i]))
# Example IHC image
for i in idx:
    null = np.zeros_like(ihc_hed[i][:, :, 0])
    ihc_h=hed2rgb(np.stack((ihc_hed[i][:, :, 0], null, null), axis=-1))
#     # Display
    fig, axes = plt.subplots(1, 2, figsize=(7, 6), sharex=True, sharey=True)
```

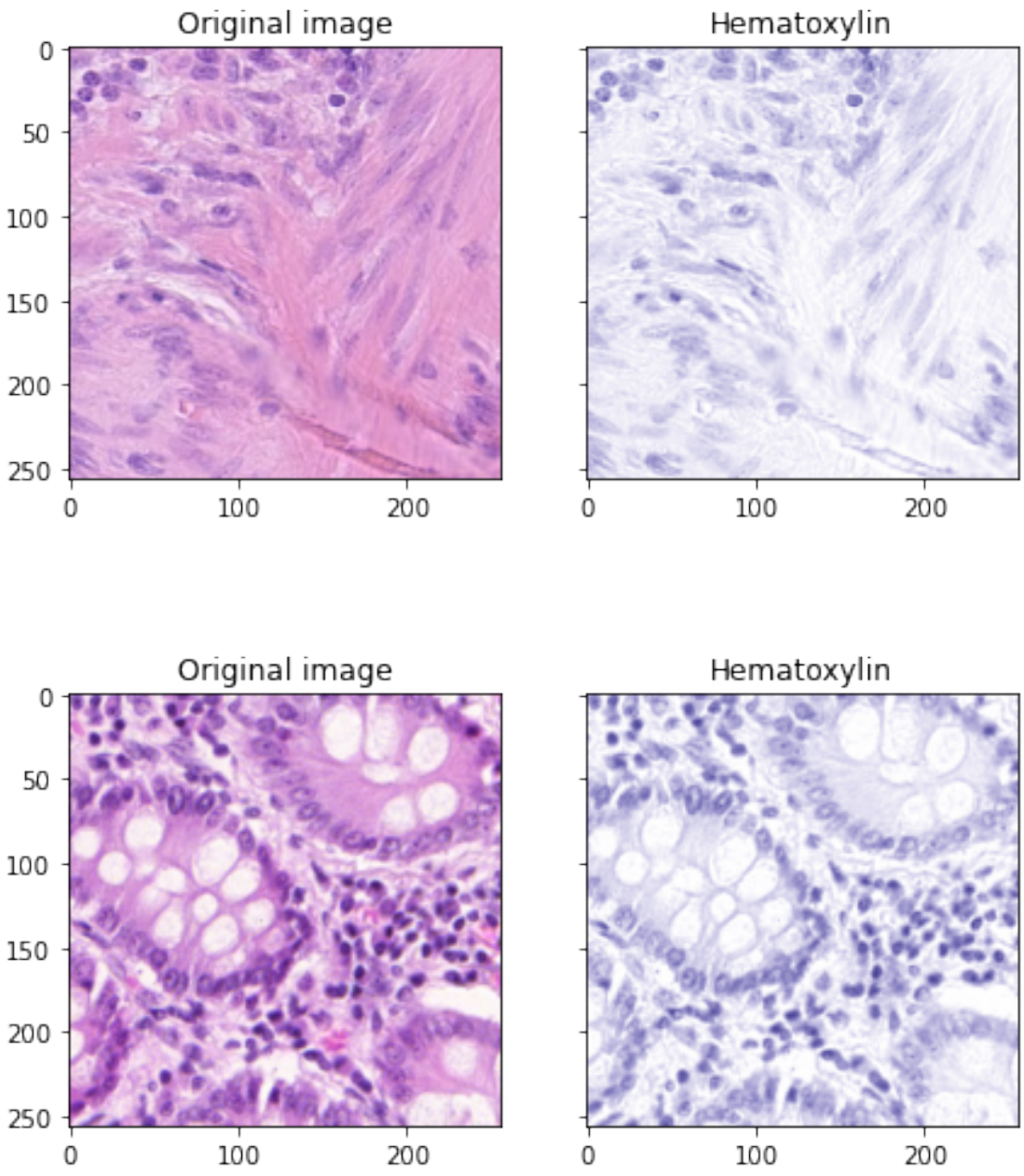
```
    ax = axes.ravel()

    ax[0].imshow(X[i])
    ax[0].set_title("Original image")


    ax[1].imshow(ihc_h)
    ax[1].set_title("Hematoxylin")
```




Original image
Hematoxylin
Original image
Hematoxylin
0
50
100
150
200
250
0
100
200


Original image
Hematoxylin
Original image
Hematoxylin
0
50
100
150
200
250
0
100
200


Original image
Hematoxylin
Original image
Hematoxylin
0
50
100
150
200
250
0
100
200



1.0.5 v. Do a scatter plot of the average of the H-channel for each image vs. its cell count of a certain type and the total number of cells for images in Fold-1 (7 plots in total). Do you think this feature would be useful in your regression model? Explain your reasoning.

I think that in a regression model, the mean of the H-channel of the total number of cells would be useful. However, there is no significant linear correlation between the mean of the H-channel for a particular type of cell number and the number of cells. In the first, third, fourth and seven scatter plots we can see a positive and linear correlation between the mean H-channel value and the total

number of cells, but we may need to transform the data, for example by taking the logarithm, to make it easier to interpret. Other images show that possibly performing a transformation would make the relationship between the H-channel mean and the number of cells a little more obvious.

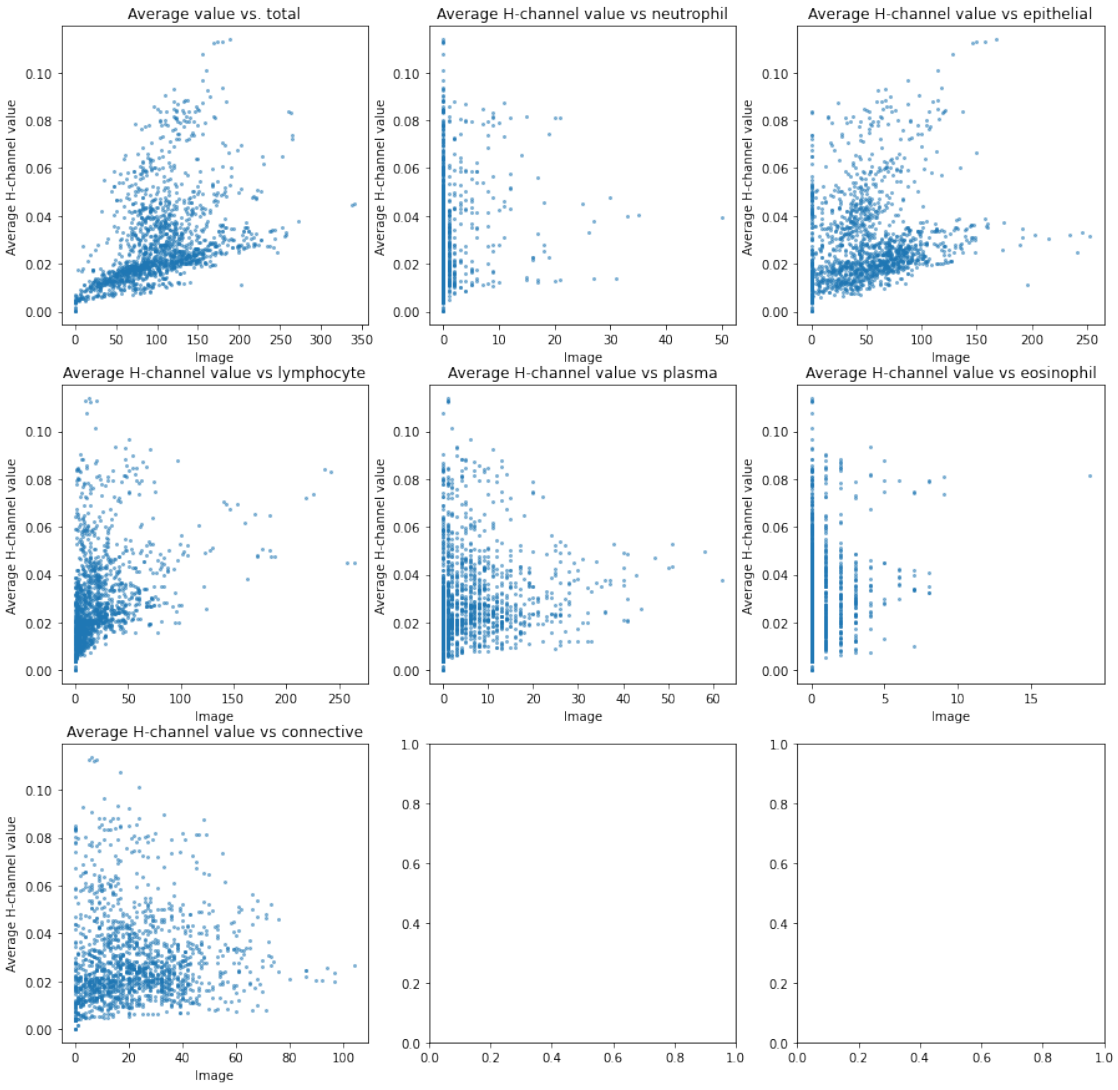
[11]:
```
np.random.randint(X[F==1].shape[0],size=10)
```

[11]:
```
array([1204,  562, 1270, 1010, 1350,  918,   96,  711,  213,  186])
```

[127]:
```
neutrophil=Y_fold1.loc[:,'neutrophil'].tolist()
epithelial=Y_fold1.loc[:,'epithelial'].tolist()
lymphocyte=Y_fold1.loc[:,'lymphocyte'].tolist()
plasma=Y_fold1.loc[:,'plasma'].tolist()
eosinophil=Y_fold1.loc[:,'eosinophil'].tolist()
connective=Y_fold1.loc[:,'connective'].tolist()

rgb_fold1= X[F==1]
fold1_hed=rgb2hed(rgb_fold1)
fold1_h=fold1_hed[:,:,:,0]
h_avg=np.mean(fold1_h,axis=(1,2))

fig, axs = plt.subplots(3, 3, figsize=(15,15))


axs[0,0].scatter(To_Y_fold1,h_avg,s=4,alpha=0.5)
axs[0,0].set_xlabel('Image')
axs[0,0].set_ylabel('Average H-channel value')
axs[0,0].set_title('Average value vs. total')

axs[0,1].scatter(neutrophil,h_avg,s=4,alpha=0.5)
axs[0,1].set_xlabel('Image')
axs[0,1].set_ylabel('Average H-channel value')
axs[0,1].set_title('Average H-channel value vs neutrophil')

axs[0,2].scatter(epithelial,h_avg,s=4,alpha=0.5)
axs[0,2].set_xlabel('Image')
axs[0,2].set_ylabel('Average H-channel value')
axs[0,2].set_title('Average H-channel value vs epithelial')

axs[1,0].scatter(lymphocyte,h_avg,s=4,alpha=0.5)
axs[1,0].set_xlabel('Image')
axs[1,0].set_ylabel('Average H-channel value')
axs[1,0].set_title('Average H-channel value vs lymphocyte')

axs[1,1].scatter(plasma,h_avg,s=4,alpha=0.5)
axs[1,1].set_xlabel('Image')
axs[1,1].set_ylabel('Average H-channel value')
axs[1,1].set_title('Average H-channel value vs plasma')
```

```
axs[1,2].scatter(eosinophil,h_avg,s=4,alpha=0.5)
axs[1,2].set_xlabel('Image')
axs[1,2].set_ylabel('Average H-channel value')
axs[1,2].set_title('Average H-channel value vs eosinophil')

axs[2,0].scatter(connective,h_avg,s=4,alpha=0.5)
axs[2,0].set_xlabel('Image')
axs[2,0].set_ylabel('Average H-channel value')
axs[2,0].set_title('Average H-channel value vs connective')
```

[127]: Text(0.5, 1.0, 'Average H-channel value vs connective')



[ ]:

1.0.6 vi. What performance metrics can you use for this problem? Which one will be the best performance metric for this problem? Please give your reasoning.

The choice of the best performance metric for regression problems depends on the specific objective of the regression model. If the aim is to make accurate predictions of the total number of cells in an image, then metrics like Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) may be appropriate since they penalize larger errors more heavily, which is crucial when minimizing the overall prediction error.

In regression problems, other metrics are commonly used to assess the accuracy of predicted continuous values, with Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2) being the most frequently used ones.

Alternatively, if the objective is to predict the relative cell count of a particular type, then metrics such as Mean Absolute Error (MAE) or Mean Absolute Percentage Error (MAPE) may be more suitable since they give equal importance to all errors, which is crucial when accurately predicting the proportion of a particular cell type.

Ultimately, the choice of the best performance metric will depend on the specific objectives of the regression model and the context in which it will be applied.

2 Question No. 2: (Feature Extraction and Classical Regression)

2.0.1 i. Extract features from a given image. Specifically, calculate the:

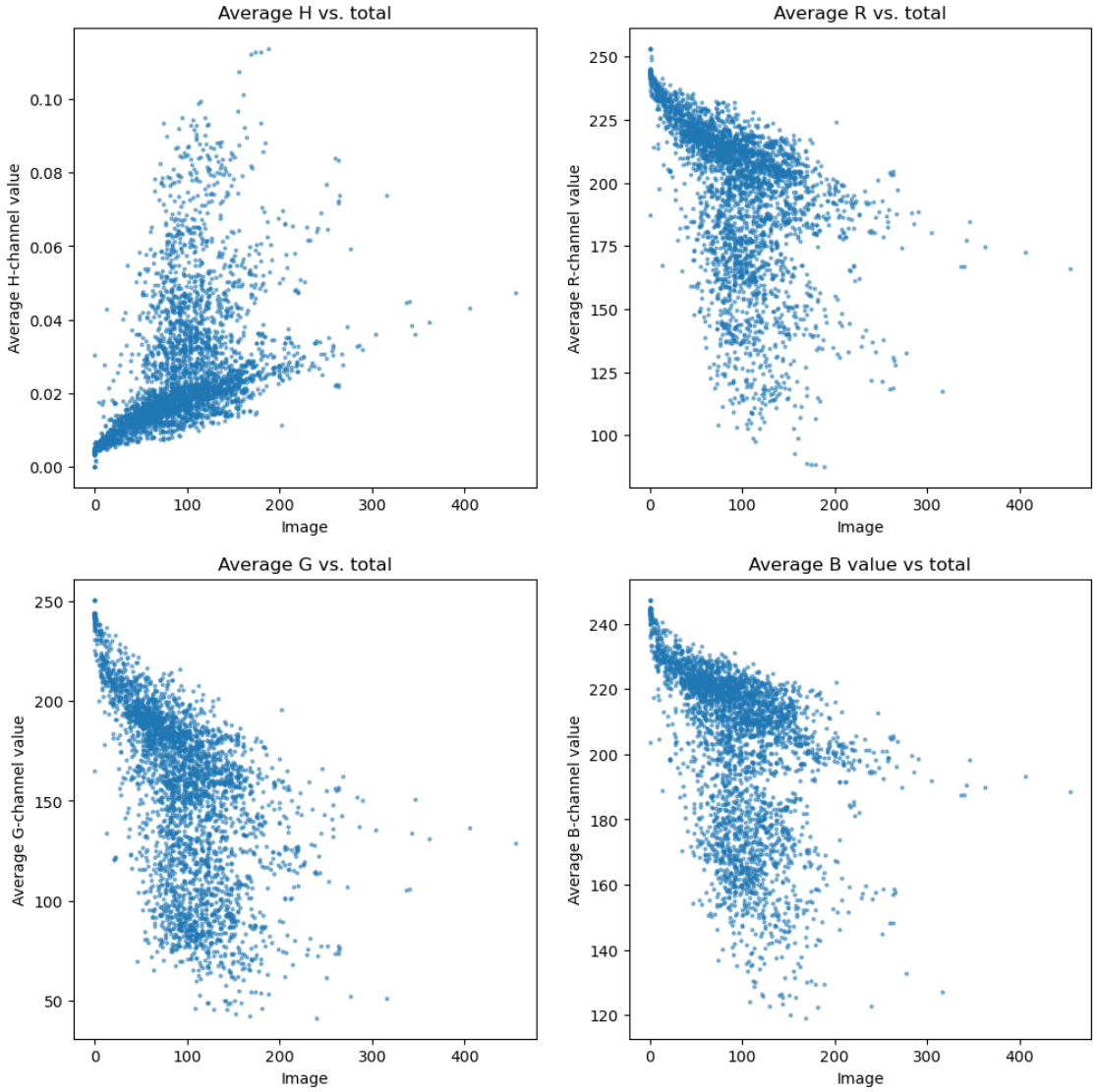
2.0.2 a. average of the "H", red, green and blue channels

[3]:
```
def avg_H_R_G_B(dataset_rgb):
    r_avg=[]
    g_avg=[]
    b_avg=[]
    fold_hed=rgb2hed(dataset_rgb)
    fold_h=fold_hed[:,:,:,0]
    h_avg=np.mean(fold_h,axis=(1,2))
    for i in range(len(dataset_rgb)):
        # Extract the H, R, G, and B channels
        r_channel = dataset_rgb[i][:, :, 0]
        g_channel = dataset_rgb[i][:, :, 1]
        b_channel = dataset_rgb[i][:, :, 2]

        # Calculate the average values of each channel
        r_avg.append(np.mean(r_channel))
        g_avg.append(np.mean(g_channel))
        b_avg.append(np.mean(b_channel))
    return h_avg, r_avg, g_avg, b_avg
```

[4]:
```
h_avg, r_avg, g_avg, b_avg=avg_H_R_G_B(dataset_rgb=X[(F==1)|(F==2)])
```

[5]:
```
Y_fold12=Y[(F==1)|(F==2)]
To_Y_fold12=Y_fold12.apply(lambda x:x.sum(),axis =1)
# scatter plot
fig, axs = plt.subplots(2, 2, figsize=(12,12))

axs[0,0].scatter(To_Y_fold12,h_avg,s=4,alpha=0.5)
axs[0,0].set_xlabel('Image')
axs[0,0].set_ylabel('Average H-channel value')
axs[0,0].set_title('Average H vs. total')

axs[0,1].scatter(To_Y_fold12,r_avg,s=4,alpha=0.5)
axs[0,1].set_xlabel('Image')
axs[0,1].set_ylabel('Average R-channel value')
axs[0,1].set_title('Average R vs. total')

axs[1,0].scatter(To_Y_fold12,g_avg,s=4,alpha=0.5)
axs[1,0].set_xlabel('Image')
axs[1,0].set_ylabel('Average G-channel value')
axs[1,0].set_title('Average G vs. total')

axs[1,1].scatter(To_Y_fold12,b_avg,s=4,alpha=0.5)
axs[1,1].set_xlabel('Image')
axs[1,1].set_ylabel('Average B-channel value')
axs[1,1].set_title('Average B value vs total')

print('The correlation coefficient of average H-channel and cell count is',␣
 ↪str(round(np.corrcoef(h_avg, To_Y_fold12)[0][1],5)))
print('The correlation coefficient of average R-channel and cell count is',␣
 ↪str(round(np.corrcoef(r_avg, To_Y_fold12)[0][1],5)))
print('The correlation coefficient of average G-channel and cell count is',␣
 ↪str(round(np.corrcoef(g_avg, To_Y_fold12)[0][1],5)))
print('The correlation coefficient of average B-channel and cell count is',␣
 ↪str(round(np.corrcoef(b_avg, To_Y_fold12)[0][1],5)))
```

```
The correlation coefficient of average H-channel and cell count is 0.43365
The correlation coefficient of average R-channel and cell count is -0.48831
The correlation coefficient of average G-channel and cell count is -0.58009
The correlation coefficient of average B-channel and cell count is -0.46792
```



2.0.3 b. variance of the "H", red, green and blue channels

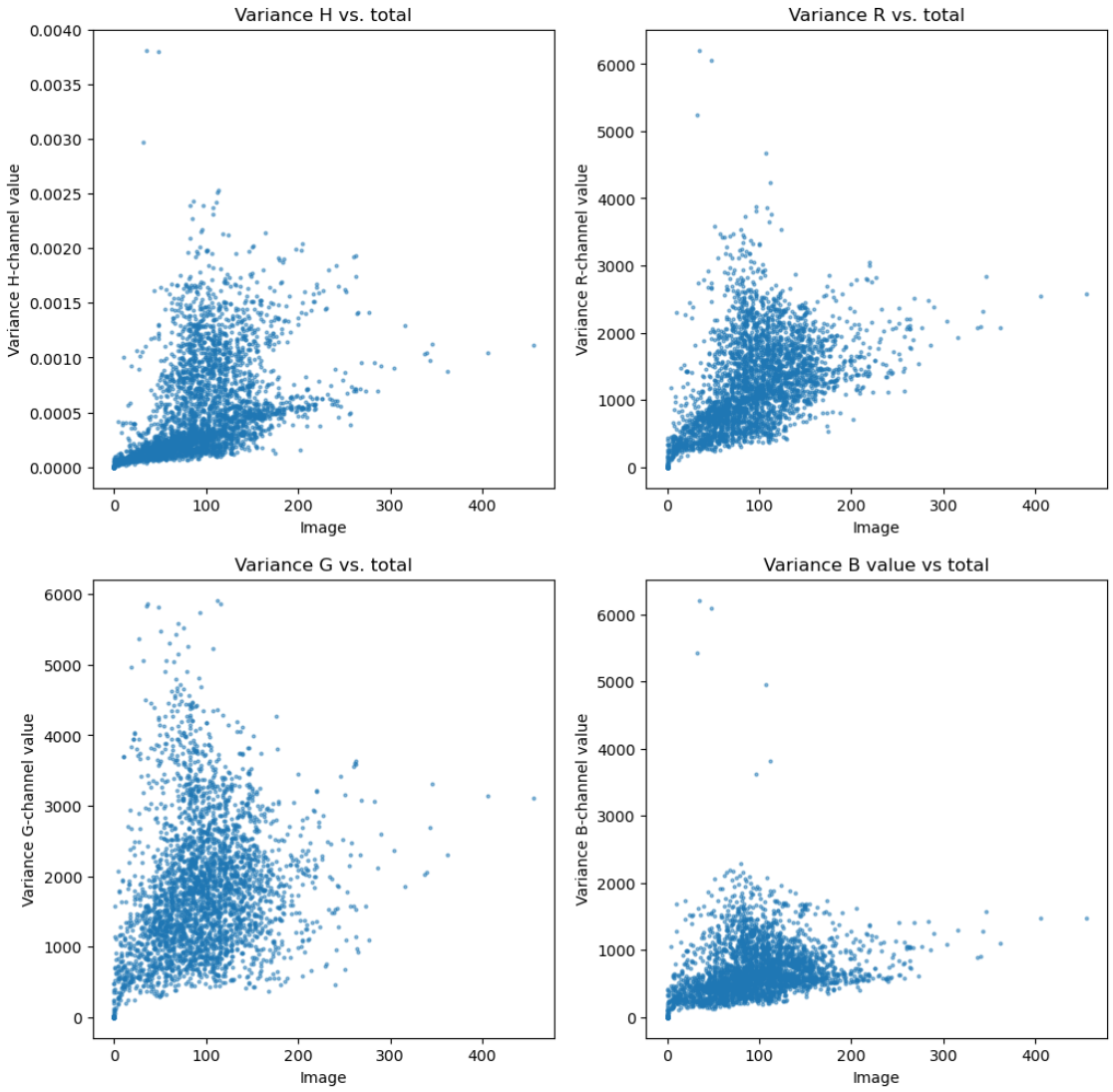
[6]:
```
def var_H_R_G_B(dataset_rgb):
    r_var=[]
    g_var=[]
    b_var=[]
    fold_hed=rgb2hed(dataset_rgb)
    fold_h=fold_hed[:,:,:,0]
    h_var=np.var(fold_h,axis=(1,2))
    for i in range(len(dataset_rgb)):
        # Extract the H, R, G, and B channels
#         h_channel = dataset_H[i][:, :, 0]
        r_channel = dataset_rgb[i][:, :, 0]
```

```
        g_channel = dataset_rgb[i][:, :, 1]
        b_channel = dataset_rgb[i][:, :, 2]
        # Calculate the average values of each channel
#         h_var.append(np.var(h_channel))
        r_var.append(np.var(r_channel))
        g_var.append(np.var(g_channel))
        b_var.append(np.var(b_channel))
    return h_var, r_var, g_var, b_var
```

[7]:
```
h_var, r_var, g_var, b_var=var_H_R_G_B(dataset_rgb=X[(F==1)|(F==2)])

# scatter plot
fig, axs = plt.subplots(2, 2, figsize=(12,12))


axs[0,0].scatter(To_Y_fold12,h_var,s=4,alpha=0.5)
axs[0,0].set_xlabel('Image')
axs[0,0].set_ylabel('Variance H-channel value')
axs[0,0].set_title('Variance H vs. total')

axs[0,1].scatter(To_Y_fold12,r_var,s=4,alpha=0.5)
axs[0,1].set_xlabel('Image')
axs[0,1].set_ylabel('Variance R-channel value')
axs[0,1].set_title('Variance R vs. total')

axs[1,0].scatter(To_Y_fold12,g_var,s=4,alpha=0.5)
axs[1,0].set_xlabel('Image')
axs[1,0].set_ylabel('Variance G-channel value')
axs[1,0].set_title('Variance G vs. total')

axs[1,1].scatter(To_Y_fold12,b_var,s=4,alpha=0.5)
axs[1,1].set_xlabel('Image')
axs[1,1].set_ylabel('Variance B-channel value')
axs[1,1].set_title('Variance B value vs total')

print('The correlation coefficient of variance H-channel and cell count is',␣
 ↪str(round(np.corrcoef(h_var, To_Y_fold12)[0][1],5)))
print('The correlation coefficient of variance R-channel and cell count is',␣
 ↪str(round(np.corrcoef(r_var, To_Y_fold12)[0][1],5)))
print('The correlation coefficient of variance G-channel and cell count is',␣
 ↪str(round(np.corrcoef(g_var, To_Y_fold12)[0][1],5)))
print('The correlation coefficient of variance B-channel and cell count is',␣
 ↪str(round(np.corrcoef(b_var, To_Y_fold12)[0][1],5)))
```

```
The correlation coefficient of variance H-channel and cell count is 0.42902
The correlation coefficient of variance R-channel and cell count is 0.48163
The correlation coefficient of variance G-channel and cell count is 0.2405
```

```
The correlation coefficient of variance B-channel and cell count is 0.28622
```



2.0.4 c. Any other features that you think can be useful for this work. Describe your reasoning for using these features. HINT/Suggestion: You may want to use PCA Coefficients of image data (you may want to use randomized PCA or incremental PCA, see:

2.0.5 PCA

PCA is primarily used to reduce the dimensionality of a high-dimensional dataset while retaining as much of the original data information as possible. It achieves this by generating a set of principal components, which are linear combinations of the original data. The weights or coefficients of these principal components describe their contribution to the original data.

If the size of the original dataset is increased from 1000 to 2000 data points, the coefficients of the

principal components in the PCA result may change. This is because the increased dataset size may reveal more data variability that can impact the coefficients of the principal components.

However, in practical scenarios, the changes in principal component coefficients may not be significant, particularly when large datasets are used for PCA calculation, and these changes may tend to stabilize over time. Moreover, even if the coefficients undergo some changes, they can still explain the principal components' contribution to the original data.

Thus, increasing the dataset size may lead to some changes in principal component coefficients, but these changes may not significantly affect the interpretation of the principal components.

[3]:
```
# find the best number component which can explain 95% variance
def pca_component(X_fold):
    avg_component=[]

    fold_hed=rgb2hed(X_fold)
    # pca coefficients
    h_channel = fold_hed[:,:, :, 0].reshape(-1,256*256)
    # Centralize
    h_channel = h_channel-np.mean(h_channel, axis=0)
    pca_h = PCA()
    pca_h.fit(h_channel)
    avg_component.append(np.where(np.cumsum(pca_h.explained_variance_ratio_)>=0.
↪95)[0][0]+1)
    del h_channel,pca_h

    r_channel = X_fold[:,:, :, 0].reshape(-1,256*256)
    r_channel = r_channel-np.mean(r_channel, axis=0)
    pca_r = PCA()
    pca_r.fit(r_channel)
    avg_component.append(np.where(np.cumsum(pca_r.explained_variance_ratio_)>=0.
↪95)[0][0]+1)
    del r_channel,pca_r

    g_channel = X_fold[:,:, :, 1].reshape(-1,256*256)
    g_channel = g_channel-np.mean(g_channel, axis=0)
    pca_g = PCA()
    pca_g.fit(g_channel)
    avg_component.append(np.where(np.cumsum(pca_g.explained_variance_ratio_)>=0.
↪95)[0][0]+1)
    del g_channel,pca_g

    b_channel = X_fold[:,:, :, 2].reshape(-1,256*256)
    b_channel = b_channel-np.mean(b_channel, axis=0)
    pca_b = PCA()
    pca_b.fit(b_channel)
    avg_component.append(np.where(np.cumsum(pca_b.explained_variance_ratio_)>=0.
↪95)[0][0]+1)
```

```
    del b_channel,pca_b

    return int(np.mean(avg_component))
```

[4]:
```
component=pca_component(X[F==1])
component
```

[4]:
```
907
```

Due to the large size of the data, it was necessary to run it separately, and in order to avoid running out of memory and to reduce the runtime, I stored the results after running them.

[3]:
```
component=907
def pca_H_feature(X_fold,component):
    fold_hed=rgb2hed(X_fold)

    # Process H channel
    h_channel = fold_hed[:,:, :, 0].reshape(-1,256*256)
    #
    h_channel = h_channel-np.mean(h_channel, axis=0)
    pca_h = PCA(n_components=component)
    X_H_pca=pca_h.fit_transform(h_channel)
    return X_H_pca
```

[4]:
```
H_pca_train_feature=pca_H_feature(X[(F==1)| (F==2)],907)
#saving the result to CSV document.
np.savetxt('H_pca_train_feature.csv', H_pca_train_feature, delimiter=',')
del H_pca_train_feature
```

[4]:
```
def pca_rgb_feature(X_fold,rgb,component):

    r_channel = X_fold[:,:, :, rgb].reshape(-1,256*256)
    r_channel = r_channel-np.mean(r_channel, axis=0)
    pca_r = PCA(n_components=component)
    X_R_pca=pca_r.fit_transform(r_channel)

    return X_R_pca
```

[ ]:
```
r_pca_train_feature=pca_rgb_feature(X[(F==1)| (F==2)],0,component)
np.savetxt('r_pca_train_feature.csv', r_pca_train_feature, delimiter=',')
del r_pca_train_feature

g_pca_train_feature=pca_rgb_feature(X[(F==1)| (F==2)],1,component)
np.savetxt('g_pca_train_feature.csv', g_pca_train_feature, delimiter=',')
del g_pca_train_feature

b_pca_train_feature=pca_rgb_feature(X[(F==1)| (F==2)],2,component)
np.savetxt('b_pca_train_feature.csv', b_pca_train_feature, delimiter=',')
```

```
del b_pca_train_feature
```

[9]:
```
X_H_pca = np.loadtxt('H_pca_train_feature.csv', delimiter=',')
X_R_pca = np.loadtxt('r_pca_train_feature.csv', delimiter=',')
X_G_pca = np.loadtxt('g_pca_train_feature.csv', delimiter=',')
X_B_pca = np.loadtxt('b_pca_train_feature.csv', delimiter=',')
feature_pca = np.concatenate((X_H_pca, X_R_pca, X_G_pca, X_B_pca), axis=1)
feature_pca

np.savetxt('pca_train_feature.csv', feature_pca, delimiter=',')
```

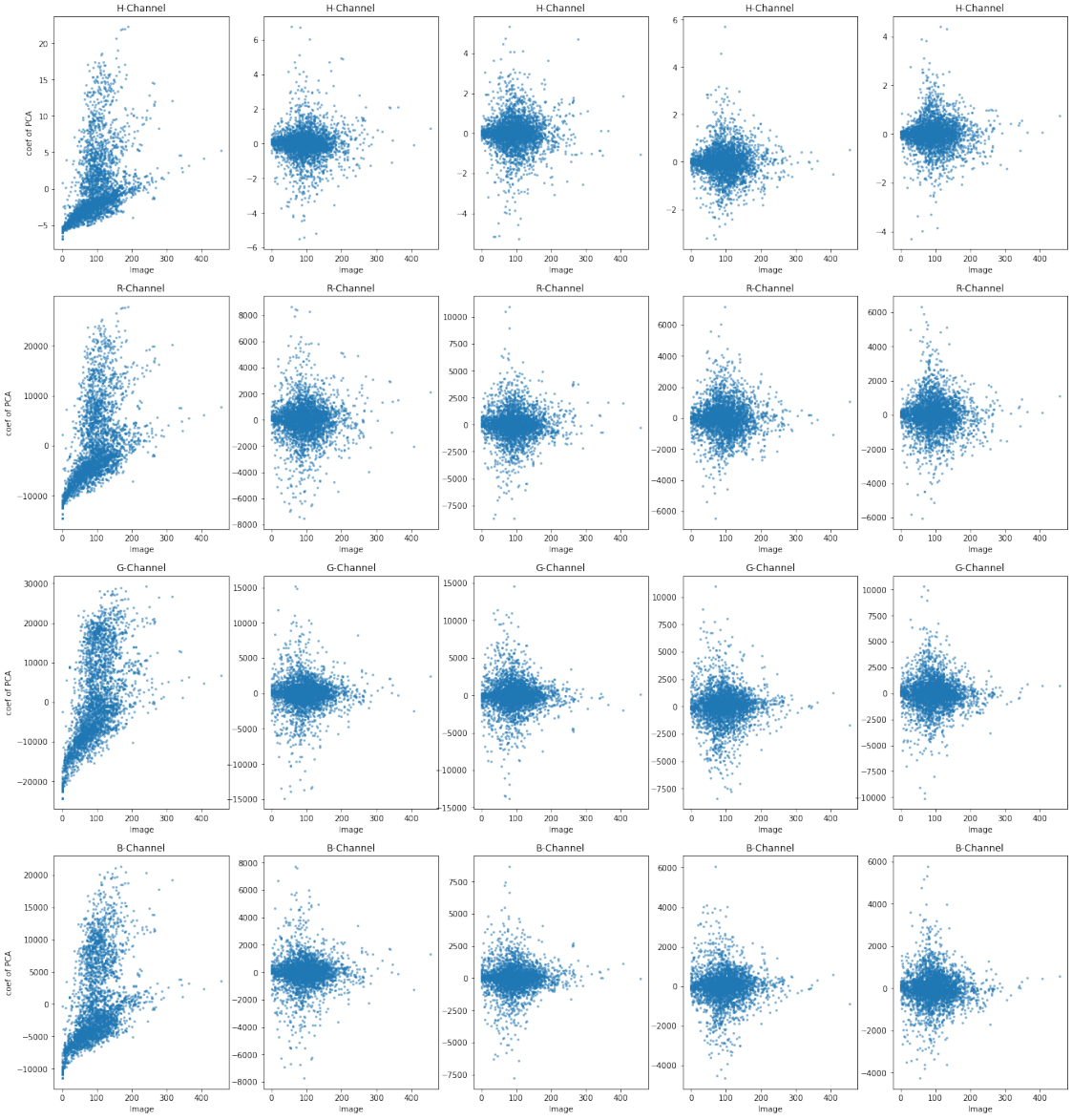
[8]:
```
feature_pca = np.loadtxt('pca_train_feature.csv', delimiter=',')
```

[9]:
```
Y_fold12=Y[(F==1)| (F==2)]
To_Y_fold12=Y_fold12.apply(lambda x:x.sum(),axis =1)
```

[16]:
```
table_coef_pca = [['␣
 ↪','PC1','PC2','PC3','PC4','PC5','PC6','PC7','PC8','PC9','PC10']]
channel=['H-Channel','R-Channel','G-Channel','B-Channel']
# scatter plot
fig, axs = plt.subplots(4, 5,figsize=(23,25))
#, figsize=(12,12)
for j in range(len(channel)):
    dif_channel=component*j
    new=[[channel[j],
          round(np.corrcoef(feature_pca[:,dif_channel],To_Y_fold12)[0][1],5),
          round(np.corrcoef(feature_pca[:,dif_channel+1],To_Y_fold12)[0][1],5),
          round(np.corrcoef(feature_pca[:,dif_channel+2],To_Y_fold12)[0][1],5),
          round(np.corrcoef(feature_pca[:,dif_channel+3],To_Y_fold12)[0][1],5),
          round(np.corrcoef(feature_pca[:,dif_channel+4],To_Y_fold12)[0][1],5),
          round(np.corrcoef(feature_pca[:,dif_channel+5],To_Y_fold12)[0][1],5),
          round(np.corrcoef(feature_pca[:,dif_channel+6],To_Y_fold12)[0][1],5),
          round(np.corrcoef(feature_pca[:,dif_channel+7],To_Y_fold12)[0][1],5),
          round(np.corrcoef(feature_pca[:,dif_channel+8],To_Y_fold12)[0][1],5),
          round(np.corrcoef(feature_pca[:,dif_channel+9],To_Y_fold12)[0][1],5)
         ]]
    table_coef_pca=np.r_[table_coef_pca,new]
    for i in range(5):
        axs[j,i].scatter(To_Y_fold12,feature_pca[:,dif_channel+i],s=4,alpha=0.5)
        axs[j,i].set_xlabel('Image')
        axs[j,0].set_ylabel('coef of PCA')
        axs[j,i].set_title(channel[j])


print(tabulate(table_coef_pca))
```

```
---------  -------  -------  --------  --------  --------  -------  --------
--------  --------  -------
```

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
PC8	PC9	PC10					
H-Channel	0.43362	0.02504	0.00541	-0.00763	-0.01065	0.02927	-0.00276
-0.00085	0.00847	0.00745					
R-Channel	0.48831	0.01601	0.02314	-0.0008	-0.01454	0.03154	0.00065
0.01088	-0.00394	0.01345					
G-Channel	0.58008	0.00857	-0.02209	0.00371	-0.01972	0.0341	-0.01082
-0.01175	0.00571	0.01236					
B-Channel	0.4679	0.01376	0.0285	-0.00471	-0.0177	0.03743	0.00771
-0.00321	-0.00139	0.0242					
---	---	---	---	---	---	---	---
---	---	---					



In summary, only the correlation coefficient between the first principal component and the total number of cells exceeds 0.2 in PCA.

2.0.6 GLCM

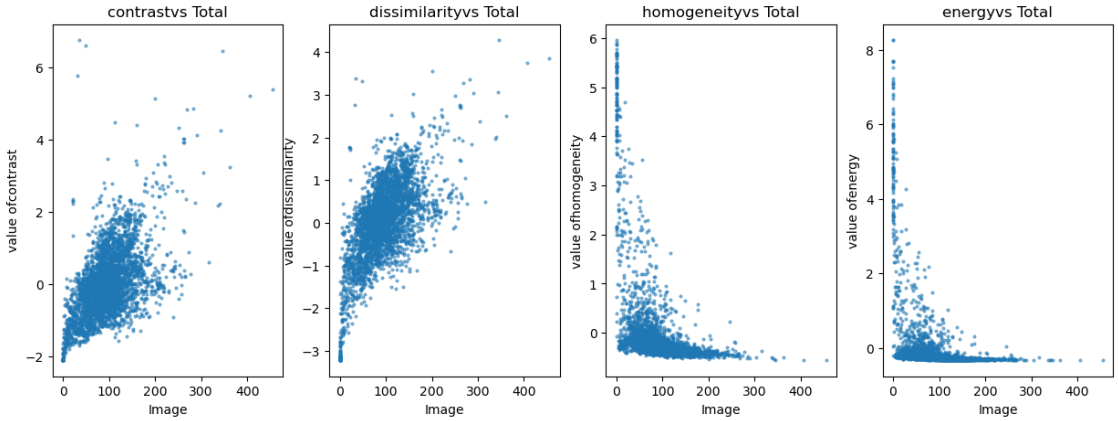
[10]:
```
def GLCM(data):
    # Converting RGB images to grayscale
    X_gray = np.array([rgb2gray(data[i]) for i in range(data.shape[0])])

    # Initialize feature vectors
    features = np.zeros((data.shape[0], 4))

    # For each image, GLCM features are calculated and stored in the feature␣
 ↪vector
    for i in range(data.shape[0]):
        glcm = greycomatrix((X_gray[i]*255).astype(np.uint8), [5], [0, np.pi/4,␣
 ↪np.pi/2, 3*np.pi/4])
        features[i, 0] = greycoprops(glcm, 'contrast').mean()
        features[i, 1] = greycoprops(glcm, 'dissimilarity').mean()
        features[i, 2] = greycoprops(glcm, 'homogeneity').mean()
        features[i, 3] = greycoprops(glcm, 'energy').mean()

    # Normalisation of feature vectors
    features = (features - features.mean(axis=0)) / features.std(axis=0)
    return features
```

[ ]:
```
glcm_feature=GLCM(X[(F==1)| (F==2)])
```

[18]:
```
table_coef_glcm = [[' ','contrast','dissimilarity','homogeneity','energy']]
# scatter plot
fig, axs = plt.subplots(1, 4,figsize=(15,5))

for i in range(4):
        axs[i].scatter(To_Y_fold12,glcm_feature[:,i],s=4,alpha=0.5)
        axs[i].set_xlabel('Image')
        axs[i].set_ylabel('value of'+table_coef_glcm[0][i+1])
        axs[i].set_title(table_coef_glcm[0][i+1]+'vs Total')


new_glcm=[['GLCM',
      round(np.corrcoef(glcm_feature[:,0],To_Y_fold12)[0][1],5),
      round(np.corrcoef(glcm_feature[:,1],To_Y_fold12)[0][1],5),
      round(np.corrcoef(glcm_feature[:,2],To_Y_fold12)[0][1],5),
      round(np.corrcoef(glcm_feature[:,3],To_Y_fold12)[0][1],5)
     ]]
table_coef_glcm=np.r_[table_coef_glcm,new_glcm]
```

```
print(tabulate(table_coef_glcm))
```

```
----  --------  -------------  -----------  --------
      contrast  dissimilarity  homogeneity  energy
GLCM  0.58061   0.65338        -0.48899     -0.41972
----  --------  -------------  -----------  --------
```



In summary, the absolute value of the correlation coefficient for each feature exceeds 0.4

2.0.7 ii. Try the following regression models with the features used in part-I. Plot the scatter plot between true and predicted counts for each type of regression model for the test data. Also, report your prediction performance in terms of RMSE, Pearson Correlation Coefficient, Spearman Correlation Coefficient and R2 score (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics) on the test data.

generate the test dataset and train dataset

[ ]:
```
H_pca_test_feature=pca_H_feature(X[F==3],907)
np.savetxt('H_pca_test_feature.csv', H_pca_test_feature, delimiter=',')
del H_pca_test_feature

r_pca_test_feature=pca_rgb_feature(X[F==3],1,907)
np.savetxt('r_pca_test_feature.csv', r_pca_test_feature, delimiter=',')
del r_pca_test_feature

g_pca_test_feature=pca_rgb_feature(X[F==3],2,907)
np.savetxt('g_pca_test_feature.csv', g_pca_test_feature, delimiter=',')
del g_pca_test_feature

b_pca_test_feature=pca_rgb_feature(X[F==3],0,907)
np.savetxt('b_pca_test_feature.csv', b_pca_test_feature, delimiter=',')
del b_pca_test_feature
```

[10]:
```
X_H_pca_test = np.loadtxt('H_pca_test_feature.csv', delimiter=',')
X_R_pca_test = np.loadtxt('r_pca_test_feature.csv', delimiter=',')
X_G_pca_test = np.loadtxt('g_pca_test_feature.csv', delimiter=',')
X_B_pca_test = np.loadtxt('b_pca_test_feature.csv', delimiter=',')
feature_pca_test = np.concatenate((X_H_pca_test, X_R_pca_test, X_G_pca_test,␣
 ↪X_B_pca_test), axis=1)
np.savetxt('feature_pca_test.csv', feature_pca_test, delimiter=',')
del X_H_pca_test,X_R_pca_test,X_G_pca_test,X_B_pca_test
```

[12]:
```
feature_pca_test = np.loadtxt('feature_pca_test.csv', delimiter=',')
```

[ ]:
```
# test dataset
component=907
h_avg_test, r_avg_test, g_avg_test, b_avg_test =␣
 ↪avg_H_R_G_B(dataset_rgb=X[F==3])
h_var_test, r_var_test, g_var_test, b_var_test=var_H_R_G_B(dataset_rgb=X[F==3])

glcm_feature_test=GLCM(X[F==3])

X_test_avg_var_pca_glcm = DataFrame(np.vstack([h_avg_test, r_avg_test,␣
 ↪g_avg_test, b_avg_test,
                    h_var_test, r_var_test, g_var_test, b_var_test,
                    glcm_feature_test[:,0],glcm_feature_test[:
 ↪,1],glcm_feature_test[:,2],glcm_feature_test[:,3]
                                          ,feature_pca_test[:
 ↪,0],feature_pca_test[:,component],feature_pca_test[:
 ↪,component*2],feature_pca_test[:,component*3]
                                                ]).T,
                    index=None,
                    columns = ['h_avg', 'r_avg', 'g_avg','b_avg','h_var',␣
 ↪'r_var', 'g_var', 'b_var',
                               ␣
 ↪'glcm_contrast','glcm_dissimilarity','glcm_homogeneity','glcm_energy'
                                ,'h_pca1','r_pca1','g_pca1','b_pca1'
                               ])
```

[15]:
```
X_test_avg_var_pca_glcm
```

[15]:

	h_avg	r_avg	g_avg	b_avg	h_var	r_var	\
0	0.028605	197.698242	170.029984	220.461472	0.000512	1331.335242	
1	0.026077	202.473419	176.373886	222.857880	0.000533	1417.460078	
2	0.029114	197.348557	170.161255	220.063202	0.000586	1539.463942	
3	0.023964	205.994827	180.647141	223.980301	0.000504	1436.195576	
4	0.033372	188.374313	154.436249	214.194946	0.000475	1233.496379	
…	…	…	…	…	…	…	
1603	0.017153	212.572479	172.512909	217.049973	0.000274	964.245205	
1604	0.020643	206.418716	165.679520	209.284439	0.000566	1766.249466	

```
1605  0.023952  202.140808  166.652145  209.097931  0.000644  2090.342844
1606  0.023798  202.249878  164.874191  208.633499  0.000661  2105.959045
1607  0.023468  203.599594  170.092133  210.553970  0.000671  2160.485748

            g_var        b_var  glcm_contrast  glcm_dissimilarity  \
0     1876.016023   481.998760       0.810735            0.965928
1     2178.975367   509.681401       0.857835            0.828788
2     2307.646238   535.724338       0.507328            0.578942
3     2438.515856   513.623590       0.575520            0.579722
4     1642.879389   537.033895       0.341959            0.632068
…           …           …              …                   …
1603  1629.919114   580.884420       0.546894            0.683091
1604  3407.630187  1057.736126       0.213481            0.095125
1605  4087.511367  1211.674858      -0.407248           -0.393793
1606  4008.991847  1247.109223       0.660789            0.373970
1607  4406.558559  1264.357805       0.001142           -0.247558

      glcm_homogeneity  glcm_energy    h_pca1       r_pca1       g_pca1  \
0            -0.439103    -0.362401 -0.373034 -4749.207900 -5509.526716
1            -0.208924    -0.183399 -1.033533 -6404.225984 -6133.879506
2             0.009299     0.063534 -0.228566 -4750.646593 -5405.057624
3            -0.002403     0.007661 -1.549227 -7497.239372 -6410.665311
4            -0.475922    -0.378003  0.861196  -738.945031 -3901.417819
…                  …            …         …           …           …
1603         -0.399490    -0.348863 -3.301641 -5396.706310 -4640.199170
1604          0.130653    -0.015457 -2.412801 -3714.663917 -2670.492359
1605          0.284078     0.079372 -1.553795 -3871.009463 -2590.235828
1606          0.327851     0.132698 -1.601612 -3419.008046 -2481.004566
1607          0.623192     0.314351 -1.698877 -4836.224280 -2991.607031

          b_pca1
0    -1421.715926
1    -2668.576156
2    -1315.154987
3    -3547.594265
4      978.760321
…           …
1603 -5238.272733
1604 -3679.925367
1605 -2550.724070
1606 -2583.563919
1607 -2960.663852

[1608 rows x 16 columns]
```

[20]:
```
np.savetxt('test.csv', X_test_avg_var_pca_glcm, delimiter=',')
```

[22]:
```
X_test_avg_var_pca_glcm=np.loadtxt('test.csv', delimiter=',')
```

2.0.8 a. Ordinary Least Squares (OLS) regression

[ ]:
```
h_var, r_var, g_var, b_var=var_H_R_G_B(dataset_rgb=X[(F==1)|(F==2)])
h_avg, r_avg, g_avg, b_avg=avg_H_R_G_B(dataset_rgb=X[(F==1)|(F==2)])
feature_pca = np.loadtxt('pca_train_feature.csv', delimiter=',')
glcm_feature=GLCM(X[(F==1)| (F==2)])

X_train_avg_var_pca_glcm=DataFrame(np.vstack([h_avg, r_avg, g_avg, b_avg,
                     h_var, r_var, g_var, b_var,
                     glcm_feature[:,0],glcm_feature[:,1],glcm_feature[:
 ↪,2],glcm_feature[:,3]
                                    ,feature_pca[:,0],feature_pca[:
 ↪,component],feature_pca[:,component*2],feature_pca[:,component*3]
                                             ]).T,
                     index=None,
                     columns = ['h_avg', 'r_avg', 'g_avg','b_avg','h_var',␣
 ↪'r_var', 'g_var', 'b_var',
                                ␣
 ↪'glcm_contrast','glcm_dissimilarity','glcm_homogeneity','glcm_energy'
                                 ,'h_pca1','r_pca1','g_pca1','b_pca1'
                                ])
```

[17]:
```
X_train_avg_var_pca_glcm
```

[17]:

	h_avg	r_avg	g_avg	b_avg	h_var	r_var	\
0	0.014908	216.828201	184.712616	220.145798	0.000166	700.185283	
1	0.018386	210.432480	175.731857	216.199799	0.000254	968.860401	
2	0.022369	203.084442	163.588150	211.209991	0.000359	1209.026744	
3	0.011857	224.299576	196.641907	225.250229	0.000221	851.487143	
4	0.005693	238.598755	228.551178	237.590729	0.000044	246.522321	
…	…	…	…	…	…	…	
3368	0.016086	208.914917	133.911880	205.878021	0.000217	961.169152	
3369	0.022559	195.649033	112.972305	199.495331	0.000215	736.531561	
3370	0.023275	196.148666	118.573761	202.451859	0.000257	777.945046	
3371	0.022506	195.817291	113.379898	199.627487	0.000218	746.081486	
3372	0.022524	195.692642	113.010788	199.491989	0.000214	730.125578	

	g_var	b_var	glcm_contrast	glcm_dissimilarity	\
0	1525.001090	464.010344	-0.697989	-0.429730	
1	1818.421676	606.176359	-0.131541	0.176947	
2	1786.483288	673.088380	0.496611	0.693211	
3	1897.752842	550.505211	-0.401345	-0.572601	
4	616.419683	224.905800	-1.474067	-1.883588	
…	…	…	…	…	
3368	3388.763735	645.759535	-0.541020	-0.500838	

```
3369  1125.877819  376.756875      -1.109030           -0.896384
3370  1029.561729  395.359560      -0.777156           -0.469580
3371  1125.390361  377.322553      -1.117833           -0.900360
3372  1098.273519  371.541318      -1.121249           -0.910274

      glcm_homogeneity  glcm_energy    h_pca1        r_pca1        g_pca1  \
0             -0.289784    -0.271929 -3.082497  -5262.495331  -7467.186806
1             -0.376335    -0.293098 -2.195719  -3628.919399  -5167.760808
2             -0.429394    -0.305518 -1.188094  -1764.571364  -2079.966308
3              1.022201     1.040711 -3.856420  -7154.484611 -10470.614962
4              1.673266     1.280801 -5.436792 -10833.960645 -18697.407448
…                 …            …       …             …             …
3368          -0.104642    -0.222001 -2.759941  -3180.252887   5660.253234
3369          -0.262346    -0.260905 -1.121683    163.951473  10908.493808
3370          -0.334684    -0.273128 -0.948112     20.769895   9454.195954
3371          -0.264307    -0.261779 -1.139863    116.113199  10801.538070
3372          -0.260635    -0.260416 -1.132089    152.349591  10900.090342

          b_pca1
0    -4512.334671
1    -3508.471849
2    -2238.730017
3    -5805.932947
4    -8990.706671
…             …
3368  -819.937110
3369   769.842842
3370    -0.582189
3371   734.123672
3372   771.122571

[3373 rows x 16 columns]
```

[18]:
```
y_train = Y.apply(lambda x:x.sum(),axis =1)[(F==1)|(F==2)]
y_test = Y.apply(lambda x:x.sum(),axis =1)[F==3]

# using var and avg feature

# fit model
reg_avg_var_glcm  = LinearRegression().fit(X_train_avg_var_pca_glcm[['h_avg',␣
 ↪'r_avg', 'g_avg','b_avg','h_var', 'r_var', 'g_var', 'b_var']] , y_train)

# predict
y_pred_avg_var_glcm= reg_avg_var_glcm .
 ↪predict(X_test_avg_var_pca_glcm[['h_avg', 'r_avg', 'g_avg','b_avg','h_var',␣
 ↪'r_var', 'g_var', 'b_var']])
mse_avg_var_glcm= mean_squared_error(y_test, y_pred_avg_var_glcm)
```

```
r2_avg_var_glcm= r2_score(y_test, y_pred_avg_var_glcm)
pearson_avg_var_glcm=stats.pearsonr(y_test, y_pred_avg_var_glcm)
# performance matric
print("RMSE with avg and var: ", np.sqrt(mse_avg_var_glcm))
print("Pearson Correlation Coefficient with avg and var: ",␣
 ↪pearson_avg_var_glcm[0])
print("R2 score with avg and var: ", r2_avg_var_glcm)
print("spearmanr Correlation Coefficient with avg and var: ",␣
 ↪round(spearmanr(y_test, y_pred_avg_var_glcm)[0],5))
```

```
RMSE with avg and var:  41.98806201221627
Pearson Correlation Coefficient with avg and var:  0.6514203533364982
R2 score with avg and var:  0.2738912063355402
spearmanr Correlation Coefficient with avg and var:  0.68243
```

[19]:
```
y_train = Y.apply(lambda x:x.sum(),axis =1)[(F==1)|(F==2)]
y_test = Y.apply(lambda x:x.sum(),axis =1)[F==3]

# var and avg and glcm


reg_avg_var_pca_glcm = LinearRegression().
 ↪fit(X_train_avg_var_pca_glcm[['h_avg', 'r_avg', 'g_avg','b_avg','h_var',␣
 ↪'r_var', 'g_var', 'b_var','glcm_contrast']] , y_train)


y_pred_avg_var_pca_glcm= reg_avg_var_pca_glcm .
 ↪predict(X_test_avg_var_pca_glcm[['h_avg', 'r_avg', 'g_avg','b_avg','h_var',␣
 ↪'r_var', 'g_var', 'b_var','glcm_contrast']])
mse_avg_var_pca_glcm= mean_squared_error(y_test, y_pred_avg_var_pca_glcm)
r2_avg_var_pca_glcm= r2_score(y_test, y_pred_avg_var_pca_glcm)
pearson_avg_var_pca_glcm=stats.pearsonr(y_test, y_pred_avg_var_pca_glcm)


print("RMSE with avg, var and glcm_contrast: ", np.sqrt(mse_avg_var_pca_glcm))
print("Pearson Correlation Coefficient with avg, var and glcm_contrast: ",␣
 ↪pearson_avg_var_pca_glcm[0])
print("R2 score with avg, var and glcm_contrast: ", r2_avg_var_pca_glcm)
print("spearmanr Correlation Coefficient with avg, var and glcm_contrast:",␣
 ↪round(spearmanr(y_test, y_pred_avg_var_pca_glcm)[0],5))
```

```
RMSE with avg, var and glcm_contrast:  35.936294179379786
Pearson Correlation Coefficient with avg, var and glcm_contrast:
0.7534273222199888
R2 score with avg, var and glcm_contrast:  0.46811637977807063
spearmanr Correlation Coefficient with avg, var and glcm_contrast: 0.75762
```

[5]:
```
H_pca_val_feature=pca_H_feature(X[F==2],907)
#   CSV
np.savetxt('H_pca_val_feature.csv', H_pca_val_feature, delimiter=',')
del H_pca_val_feature
print('1')
r_pca_val_feature=pca_rgb_feature(X[F==2],1,907)
#    CSV
np.savetxt('r_pca_val_feature.csv', r_pca_val_feature, delimiter=',')
del r_pca_val_feature
print('1')
g_pca_val_feature=pca_rgb_feature(X[F==2],2,907)
#    CSV
np.savetxt('g_pca_val_feature.csv', g_pca_val_feature, delimiter=',')
del g_pca_val_feature
print('1')
b_pca_val_feature=pca_rgb_feature(X[F==2],0,907)
#    CSV
np.savetxt('b_pca_val_feature.csv', b_pca_val_feature, delimiter=',')
del b_pca_val_feature
print('1')
X_H_pca_val = np.loadtxt('H_pca_val_feature.csv', delimiter=',')
X_R_pca_val = np.loadtxt('r_pca_val_feature.csv', delimiter=',')
X_G_pca_val = np.loadtxt('g_pca_val_feature.csv', delimiter=',')
X_B_pca_val = np.loadtxt('b_pca_val_feature.csv', delimiter=',')
feature_pca_val = np.concatenate((X_H_pca_val, X_R_pca_val, X_G_pca_val,␣
 ↪X_B_pca_val), axis=1)

np.savetxt('feature_pca_val.csv', feature_pca_val, delimiter=',')
del X_H_pca_val,X_R_pca_val,X_G_pca_val,X_B_pca_val
```

```
1
1
1
1
```

[20]:
```
feature_pca_val = np.loadtxt('feature_pca_val.csv', delimiter=',')
```

[ ]:
```
h_avg_val, r_avg_val, g_avg_val, b_avg_val = avg_H_R_G_B(dataset_rgb=X[F==2])
h_var_val, r_var_val, g_var_val, b_var_val=var_H_R_G_B(dataset_rgb=X[F==2])


glcm_feature_val=GLCM(X[F==2])



X_val_avg_var_pca_glcm = DataFrame(np.vstack([h_avg_val, r_avg_val, g_avg_val,␣
 ↪b_avg_val,
```

```
                    h_var_val, r_var_val, g_var_val, b_var_val,
                    glcm_feature_val[:,0],glcm_feature_val[:
↪,1],glcm_feature_val[:,2],glcm_feature_val[:,3]
                                          ,feature_pca_val[:,0],feature_pca_val[:
↪,component],feature_pca_val[:,component*2],feature_pca_val[:,component*3]
                                             ]).T,
                    index=None,
                    columns = ['h_avg', 'r_avg', 'g_avg','b_avg','h_var',␣
↪'r_var', 'g_var', 'b_var',
                                ␣
↪'glcm_contrast','glcm_dissimilarity','glcm_homogeneity','glcm_energy'
                                ,'h_pca1','r_pca1','g_pca1','b_pca1'
                               ])
```

[22]:
```
X_val_avg_var_pca_glcm
```

[22]:

	h_avg	r_avg	g_avg	b_avg	h_var	r_var
0	0.020428	207.295578	177.023788	209.745300	0.000538	1835.419698
1	0.021255	204.651840	168.006271	208.284653	0.000468	1449.368973
2	0.016852	213.455811	181.105698	214.492783	0.000404	1351.265961
3	0.013158	220.426193	191.272064	219.586090	0.000258	1042.492660
4	0.009853	228.749435	210.292862	228.040482	0.000197	754.511482
…	…	…	…	…	…	…
1746	0.016086	208.914917	133.911880	205.878021	0.000217	961.169152
1747	0.022559	195.649033	112.972305	199.495331	0.000215	736.531561
1748	0.023275	196.148666	118.573761	202.451859	0.000257	777.945046
1749	0.022506	195.817291	113.379898	199.627487	0.000218	746.081486
1750	0.022524	195.692642	113.010788	199.491989	0.000214	730.125578

	g_var	b_var	glcm_contrast	glcm_dissimilarity
0	3150.091399	1406.434457	0.434938	0.258860
1	2410.381873	845.319898	0.610392	0.515831
2	2694.068921	903.908822	0.269018	0.158243
3	2532.996690	807.417607	-0.474493	-0.555833
4	1890.944734	649.620813	-0.810706	-1.129172
…	…	…	…	…
1746	3388.763735	645.759535	-0.590302	-0.531446
1747	1125.877819	376.756875	-1.172016	-0.929111
1748	1029.561729	395.359560	-0.832135	-0.500021
1749	1125.390361	377.322553	-1.181031	-0.933108
1750	1098.273519	371.541318	-1.184529	-0.943076

	glcm_homogeneity	glcm_energy	h_pca1	r_pca1	g_pca1
0	0.027366	-0.060358	-1.266791	-4642.081942	-1617.212158
1	-0.203913	-0.234798	-1.045006	-2340.828035	-1244.298428
2	-0.091942	-0.161125	-2.186115	-5712.999430	-2842.229628
3	0.587690	0.452193	-3.126707	-8350.767357	-4160.333413

```
4              1.055033     0.783435 -3.972507 -13223.036973 -6330.177088
…                 …           …        …            …           …
1746          -0.120646    -0.240584 -2.366977   6469.094913  -611.331310
1747          -0.272232    -0.279789 -0.725243  11719.280705   986.270563
1748          -0.341763    -0.292106 -0.537483  10264.584980   215.092460
1749          -0.274117    -0.280670 -0.742682  11608.594374   949.656735
1750          -0.270588    -0.279295 -0.733768  11712.331682   989.204175

           b_pca1
0    -2180.936716
1    -1487.514716
2    -3759.758061
3    -5553.234349
4    -7680.139381
…             …
1746 -2562.332124
1747   787.500642
1748   658.451040
1749   738.858270
1750   777.475722

[1751 rows x 16 columns]
```

2.0.9 b. Support Vector Regression OR Multilayer Perceptron (MLP) OR Both

[23]:
```
a=F[(F==1)|(F==2)]
```

[24]:
```
X_fold1_avg_var=X_train_avg_var_pca_glcm[a==1]
```

[25]:
```
X_fold1_avg_var_glcm=X_fold1_avg_var.iloc[:,0:12]
X_fold1_avg_var_glcm
```

[25]:
```
         h_avg       r_avg       g_avg       b_avg     h_var        r_var  \
0     0.014908  216.828201  184.712616  220.145798  0.000166   700.185283
1     0.018386  210.432480  175.731857  216.199799  0.000254   968.860401
2     0.022369  203.084442  163.588150  211.209991  0.000359  1209.026744
3     0.011857  224.299576  196.641907  225.250229  0.000221   851.487143
4     0.005693  238.598755  228.551178  237.590729  0.000044   246.522321
…        …         …           …           …          …          …
3344  0.023362  203.515259  160.305359  210.075638  0.000723  2019.279461
3345  0.023167  204.053421  159.787094  210.195587  0.000743  2078.308105
3346  0.011381  225.166473  183.013504  223.782059  0.000336  1028.205349
3347  0.022770  204.530136  159.652847  210.359131  0.000719  2019.715187
3348  0.023147  204.041397  159.629395  210.165939  0.000739  2071.619945

            g_var       b_var  glcm_contrast  glcm_dissimilarity  \
0     1525.001090  464.010344      -0.697989           -0.429730
```

```
1     1818.421676  606.176359      -0.131541            0.176947
2     1786.483288  673.088380       0.496611            0.693211
3     1897.752842  550.505211      -0.401345           -0.572601
4      616.419683  224.905800      -1.474067           -1.883588
…           …         …              …                   …
3344  3748.873522  844.821534       1.030406            0.823593
3345  3825.836827  878.677139       1.950368            1.470824
3346  2142.396073  471.881960       0.256597            0.208119
3347  3733.226027  851.168297       1.883133            1.425582
3348  3819.385631  877.553626       1.933327            1.460801

      glcm_homogeneity  glcm_energy
0            -0.289784    -0.271929
1            -0.376335    -0.293098
2            -0.429394    -0.305518
3             1.022201     1.040711
4             1.673266     1.280801
…                …           …
3344         -0.343453    -0.294803
3345         -0.388312    -0.303971
3346         -0.335531    -0.285099
3347         -0.386343    -0.304430
3348         -0.388660    -0.304377

[1622 rows x 12 columns]
```

[26]:
```
X_val_avg_var_glcm=X_val_avg_var_pca_glcm.iloc[:,0:12]
X_val_avg_var_glcm
```

[26]:
```
         h_avg       r_avg       g_avg       b_avg     h_var        r_var  \
0     0.020428  207.295578  177.023788  209.745300  0.000538  1835.419698
1     0.021255  204.651840  168.006271  208.284653  0.000468  1449.368973
2     0.016852  213.455811  181.105698  214.492783  0.000404  1351.265961
3     0.013158  220.426193  191.272064  219.586090  0.000258  1042.492660
4     0.009853  228.749435  210.292862  228.040482  0.000197   754.511482
…        …         …           …           …         …          …
1746  0.016086  208.914917  133.911880  205.878021  0.000217   961.169152
1747  0.022559  195.649033  112.972305  199.495331  0.000215   736.531561
1748  0.023275  196.148666  118.573761  202.451859  0.000257   777.945046
1749  0.022506  195.817291  113.379898  199.627487  0.000218   746.081486
1750  0.022524  195.692642  113.010788  199.491989  0.000214   730.125578

            g_var        b_var  glcm_contrast  glcm_dissimilarity  \
0     3150.091399  1406.434457       0.434938            0.258860
1     2410.381873   845.319898       0.610392            0.515831
2     2694.068921   903.908822       0.269018            0.158243
3     2532.996690   807.417607      -0.474493           -0.555833
```

```
4     1890.944734   649.620813      -0.810706           -1.129172
…           …          …              …                   …
1746  3388.763735   645.759535      -0.590302           -0.531446
1747  1125.877819   376.756875      -1.172016           -0.929111
1748  1029.561729   395.359560      -0.832135           -0.500021
1749  1125.390361   377.322553      -1.181031           -0.933108
1750  1098.273519   371.541318      -1.184529           -0.943076

      glcm_homogeneity  glcm_energy
0             0.027366    -0.060358
1            -0.203913    -0.234798
2            -0.091942    -0.161125
3             0.587690     0.452193
4             1.055033     0.783435
…                …           …
1746         -0.120646    -0.240584
1747         -0.272232    -0.279789
1748         -0.341763    -0.292106
1749         -0.274117    -0.280670
1750         -0.270588    -0.279295

[1751 rows x 12 columns]
```

[27]:
```
X_test_avg_var_glcm=X_test_avg_var_pca_glcm.iloc[:,0:12]
X_test_avg_var_glcm
```

[27]:
```
         h_avg       r_avg       g_avg       b_avg     h_var        r_var  \
0     0.028605  197.698242  170.029984  220.461472  0.000512  1331.335242
1     0.026077  202.473419  176.373886  222.857880  0.000533  1417.460078
2     0.029114  197.348557  170.161255  220.063202  0.000586  1539.463942
3     0.023964  205.994827  180.647141  223.980301  0.000504  1436.195576
4     0.033372  188.374313  154.436249  214.194946  0.000475  1233.496379
…          …           …           …           …         …           …
1603  0.017153  212.572479  172.512909  217.049973  0.000274   964.245205
1604  0.020643  206.418716  165.679520  209.284439  0.000566  1766.249466
1605  0.023952  202.140808  166.652145  209.097931  0.000644  2090.342844
1606  0.023798  202.249878  164.874191  208.633499  0.000661  2105.959045
1607  0.023468  203.599594  170.092133  210.553970  0.000671  2160.485748

            g_var        b_var  glcm_contrast  glcm_dissimilarity  \
0     1876.016023   481.998760       0.810735            0.965928
1     2178.975367   509.681401       0.857835            0.828788
2     2307.646238   535.724338       0.507328            0.578942
3     2438.515856   513.623590       0.575520            0.579722
4     1642.879389   537.033895       0.341959            0.632068
…           …           …              …                   …
1603  1629.919114   580.884420       0.546894            0.683091
```

```
1604  3407.630187  1057.736126       0.213481            0.095125
1605  4087.511367  1211.674858      -0.407248           -0.393793
1606  4008.991847  1247.109223       0.660789            0.373970
1607  4406.558559  1264.357805       0.001142           -0.247558

      glcm_homogeneity  glcm_energy
0             -0.439103    -0.362401
1             -0.208924    -0.183399
2              0.009299     0.063534
3             -0.002403     0.007661
4             -0.475922    -0.378003
…                   …            …
1603          -0.399490    -0.348863
1604           0.130653    -0.015457
1605           0.284078     0.079372
1606           0.327851     0.132698
1607           0.623192     0.314351

[1608 rows x 12 columns]
```

[28]:
```
scaler = StandardScaler()
X_fold1_avg_var_glcm = scaler.fit_transform(X_fold1_avg_var_glcm)
X_test_avg_var_glcm = scaler.transform(X_test_avg_var_glcm)
X_val_avg_var_glcm = scaler.transform(X_val_avg_var_glcm)
```

[29]:
```
y_test = Y.apply(lambda x:x.sum(),axis =1)[F==3]
validation_Y=Y.apply(lambda x:x.sum(),axis =1)[F==2]
y_fold_train=Y.apply(lambda x:x.sum(),axis =1)[F==1]

# tuning parameter
param_grid = {'C': [300,400,500]}
svr_linear = LinearSVR(random_state=5)
grid_search_linear = GridSearchCV(svr_linear, param_grid, refit=True)
# fit model
y_linear = grid_search_linear.fit(X_val_avg_var_glcm, validation_Y)
print(y_linear.best_params_)
# svr
linearsvr = LinearSVR(**y_linear.best_params_).
 ↪fit(X_fold1_avg_var_glcm,y_fold_train)

y_pred = linearsvr.predict(X_test_avg_var_glcm)

# predict

linearsvr_rmse = round(np.sqrt(mean_squared_error(y_test, y_pred)),5)
linearsvr_pearson = round(stats.pearsonr(y_test,y_pred)[0],5)
linearsvrspearman = round(stats.spearmanr(y_test,y_pred)[0],5)
```

```
linearsvr_r2 = round(r2_score(y_test,y_pred),5)

print("RMSE : ", linearsvr_rmse)
print("R2 score: ", linearsvr_r2)
print("pearson : ", linearsvr_pearson)
print("spearman: ", linearsvrspearman)
```

```
{'C': 400}
RMSE :  40.83638
R2 score:  0.31318
pearson :  0.67202
spearman:  0.70905
```

[30]:
```
np.random.seed(90)

param_grid = {'C': [1,5,10,20,30,50,70]}
svr_linear = SVR(kernel='poly' )
# GridSearchCV
grid_search_linear = GridSearchCV(svr_linear, param_grid, refit=True)

y_linear = grid_search_linear.fit(X_val_avg_var_glcm, validation_Y)
print(y_linear.best_params_)
# svr
linearsvr = LinearSVR(**y_linear.best_params_).
 ↪fit(X_fold1_avg_var_glcm,y_fold_train)

y_pred = linearsvr.predict(X_test_avg_var_glcm)

# performance

linearsvr_rmse = round(np.sqrt(mean_squared_error(y_test, y_pred)),5)
linearsvr_pearson = round(stats.pearsonr(y_test,y_pred)[0],5)
linearsvrspearman = round(stats.spearmanr(y_test,y_pred)[0],5)
linearsvr_r2 = round(r2_score(y_test,y_pred),5)

print("RMSE : ", linearsvr_rmse)
print("R2 score: ", linearsvr_r2)
print("pearson : ", linearsvr_pearson)
print("spearman: ", linearsvrspearman)
```

```
{'C': 20}
RMSE :  38.60138
R2 score:  0.3863
pearson :  0.70174
spearman:  0.73372
```

[31]:
```
# MLP
mlp = MLPClassifier()

param_grid = {
    'hidden_layer_sizes': [(128,), (256,), (128, 64), (256, 128)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam', 'sgd'],
    'alpha': [0.0001, 0.001, 0.01]
}

grid_search = GridSearchCV(mlp, param_grid,refit=True, verbose=1, n_jobs=-1)
grid_search.fit(X_val_avg_var_glcm, validation_Y)


best_params = grid_search.best_params_
print('Best Params:', best_params)

mlp_model = MLPClassifier(**grid_search.best_params_).
 ↪fit(X_fold1_avg_var_glcm,y_fold_train)

y_pred_mlp = mlp_model.predict(X_test_avg_var_glcm)


mlp_rmse = round(np.sqrt(mean_squared_error(y_test, y_pred_mlp)),5)
mlp_pearson = round(stats.pearsonr(y_test,y_pred_mlp)[0],5)
mlpspearman = round(stats.spearmanr(y_test,y_pred_mlp)[0],5)
mlp_r2 = round(r2_score(y_test,y_pred_mlp),5)

print("RMSE : ", mlp_rmse)
print("R2 score: ", mlp_r2)
print("pearson : ", mlp_pearson)
print("spearman: ", mlpspearman)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Best Params: {'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (256,
128), 'solver': 'adam'}
RMSE :  39.18198
R2 score:  0.3677
pearson :  0.71067
spearman:  0.72201
```

3 Question No. 3 (Using Convolutional Neural Networks)

3.0.1 a. Use a convolutional neural network (in PyTorch) to solve this problem in much the same was as in part (ii) of Question (2). You are to develop an architecture of the neural network that takes an image directly as input and produces a count as the output corresponding to the total number of cells. You are free to choose any network structure as long as you can show that it gives good performance. Report your results on the test examples by plotting the scatter plot between true and predicted counts on the test data. Also, report your results interms of RMSE, Pearson Correlation Coefficient, Spearman Correlation Coefficient and R2 score. You will be evaluated on the design of your machine learning model and final performance metrics. Try to get the best test performance you can. Please include convergence plots in your submission showing how does loss change over training epochs.

[3]:
```
X_train=X[F==1]
Y_train=np.array(Y[F==1].apply(lambda x:x.sum(),axis =1))

X_val=X[F==2]
Y_val=np.array(Y[F==2].apply(lambda x:x.sum(),axis =1))

X_test=X[F==3]
Y_test=np.array(Y[F==3].apply(lambda x:x.sum(),axis =1))
```

[4]:
```
USE_CUDA = torch.cuda.is_available()
from torch.autograd import Variable
def cuda(v):
    if USE_CUDA:
        return v.cuda()
    return v
def toTensor(v,dtype = torch.float,requires_grad = False):
    return cuda(Variable(torch.tensor(v)).type(dtype).
 ↪requires_grad_(requires_grad))
def toNumpy(v):
    if USE_CUDA:
        return v.detach().cpu().numpy()
    return v.detach().numpy()
def self_loss_function(model, loss_fn, val_loader, device):
    with torch.no_grad():
        Y_shuffled, Y_preds, losses = [],[],[]
        for X, Y in val_loader:
            X = X.to(device)
            Y = Y.to(device)
            model.to(device)
            preds = model(toTensor(X.reshape(-1,3,256,256)))
            loss  = loss_fn(preds, Y)
            losses.append(loss.item())
            #Y_shuffled.append(Y)
```

```
            #Y_preds.append(preds.argmax(dim=-1))
        #Y_shuffled = torch.cat(Y_shuffled)
        #Y_preds    = torch.cat(Y_preds)
        valid_loss = torch.tensor(losses).mean()
        return valid_loss

print('Using CUDA:',USE_CUDA)
```

```
Using CUDA: True
```

[5]:
```
class MyDataset(Dataset):
    def __init__(self, data,label):
        self.data = data
        self.label=label
        self.transform = transforms.Compose([
            transforms.ToTensor()
        ])
    def __len__(self):
        return len(self.data)

    def __getitem__(self, item):
        img= self.data[item]
        img = torch.from_numpy(img).type(torch.FloatTensor)
        new_label = np.array(self.label[item])
        new_label = torch.from_numpy(new_label).type(torch.FloatTensor).
 ↪expand(1)
        return img, new_label
```

[6]:
```
def transform_data(X_train,Y_train,X_val,Y_val,X_test,Y_test):
    train = MyDataset(X_train.astype(float)/255, Y_train.astype(float))
    val = MyDataset(X_val.astype(float)/255, Y_val.astype(float))
    test= MyDataset(X_test.astype(float)/255, Y_test.astype(float))

    train_loader = DataLoader(dataset=train, batch_size=200, shuffle=True)
    val_loader=DataLoader(dataset=val, batch_size=200, shuffle=True)
    test_loader = DataLoader(dataset=test, batch_size=200, shuffle=True)
    return train_loader,val_loader,test_loader
```

[7]:
```
train_loader,val_loader,test_loader=transform_data(X_train,Y_train,X_val,Y_val,X_test,Y_test)
# Device configuration
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
# Hyper parameters
num_epochs = 50
learning_rate = 0.001
```

[10]:
```
class CellCountingCNN(nn.Module):
    def __init__(self):
        super(CellCountingCNN, self).__init__()
```

```
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2))

        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2))

        self.layer3 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2))

        self.layer4 = nn.Sequential(
            nn.Conv2d(64, 64, kernel_size=2, stride=2, padding=0),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2))


        self.fc1 = nn.Linear(64,100)
        self.fc2 = nn.Linear(100, 1)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc1(out)
        out = self.fc2(out)
        return out
```

[11]:
```
model = CellCountingCNN()
model.to(device)
# Loss and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(),␣
 ↪lr=learning_rate,weight_decay=5*(1e-4))

# Train the model
total_step = len(train_loader)
```

```
Loss_value=[]
val_Loss_value=[]
val_loss_com=5000
for epoch in range(num_epochs):
    loss_list=[]
    for  i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(toTensor(images.reshape(-1,3,256,256)))
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_list.append(loss.item())

    train_loss = torch.tensor(loss_list).mean()
    Loss_value.append(train_loss)

    print("Train Loss :{:.4f}".format(train_loss))
    val_loss=self_loss_function(model,criterion,val_loader,device)
    print("Valid Loss :{:.4f}".format(val_loss))
    val_Loss_value.append(val_loss)

    #Save the best parameters so far
    if (val_loss+train_loss)<val_loss_com:
      # Save the model checkpoint
        torch.save(model.state_dict(), 'model.ckpt')
        val_loss_com = val_loss+train_loss
```

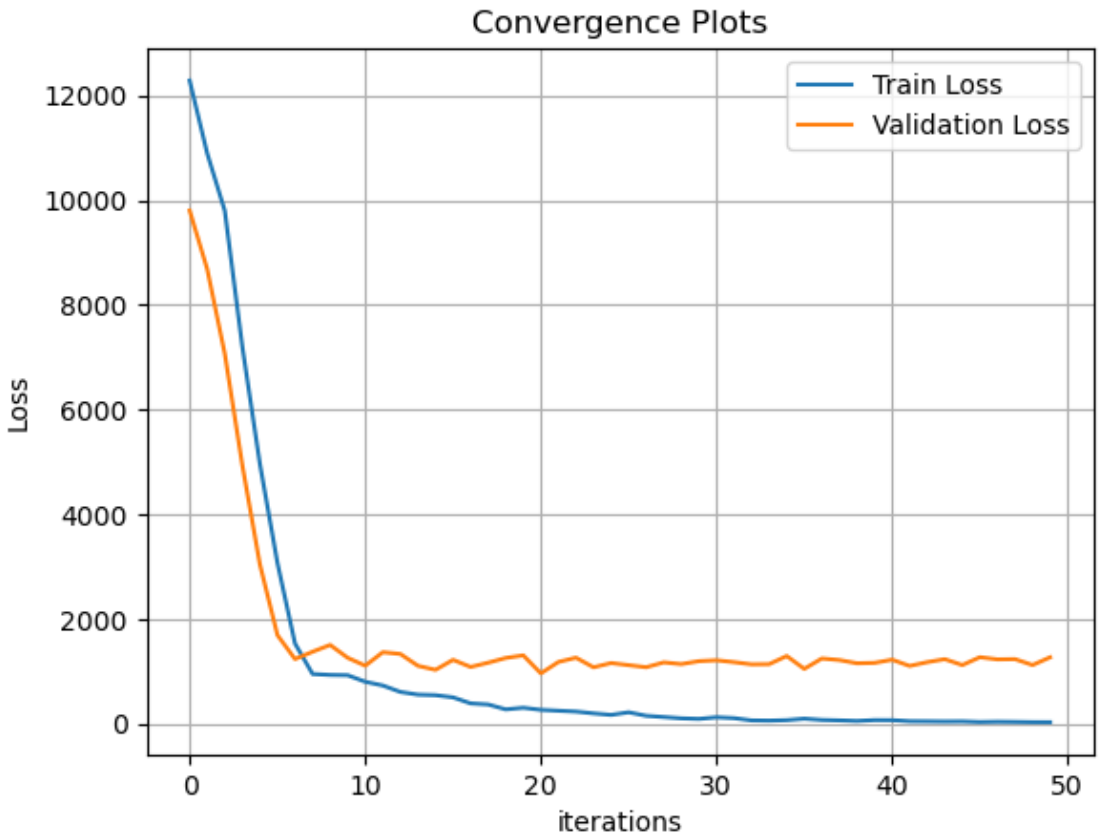
```
Train Loss :12286.9912
Valid Loss :9803.9775
Train Loss :10896.4971
Valid Loss :8685.1924
Train Loss :9801.3916
Valid Loss :7069.8506
Train Loss :7204.5718
Valid Loss :4939.9609
Train Loss :4964.3247
Valid Loss :3043.8372
Train Loss :3070.0264
Valid Loss :1691.3086
Train Loss :1531.8237
Valid Loss :1234.2434
Train Loss :948.7203
```

```
Valid Loss :1373.9131
Train Loss :930.8570
Valid Loss :1504.0717
Train Loss :925.3881
Valid Loss :1258.4585
Train Loss :800.5864
Valid Loss :1107.7836
Train Loss :728.9142
Valid Loss :1365.3619
Train Loss :606.1311
Valid Loss :1331.4905
Train Loss :554.3752
Valid Loss :1105.2587
Train Loss :544.3751
Valid Loss :1028.7798
Train Loss :502.1111
Valid Loss :1220.0305
Train Loss :387.4120
Valid Loss :1083.3373
Train Loss :366.6681
Valid Loss :1165.0359
Train Loss :272.8668
Valid Loss :1255.3258
Train Loss :303.8605
Valid Loss :1304.9406
Train Loss :263.8563
Valid Loss :961.0500
Train Loss :247.1232
Valid Loss :1179.0732
Train Loss :231.1799
Valid Loss :1260.6843
Train Loss :196.3404
Valid Loss :1076.7340
Train Loss :167.2694
Valid Loss :1158.3534
Train Loss :217.1888
Valid Loss :1117.8833
Train Loss :150.1918
Valid Loss :1078.3545
Train Loss :126.3455
Valid Loss :1170.8247
Train Loss :101.8822
Valid Loss :1138.2800
Train Loss :91.0054
Valid Loss :1194.6458
Train Loss :121.8547
Valid Loss :1211.0028
Train Loss :105.8811
```

```
Valid Loss :1175.2450
Train Loss :62.2677
Valid Loss :1133.4409
Train Loss :57.9130
Valid Loss :1136.8413
Train Loss :67.5798
Valid Loss :1292.9912
Train Loss :95.1061
Valid Loss :1046.7925
Train Loss :71.2762
Valid Loss :1245.4274
Train Loss :62.8693
Valid Loss :1215.4402
Train Loss :51.4584
Valid Loss :1153.8167
Train Loss :68.3907
Valid Loss :1161.5161
Train Loss :67.1277
Valid Loss :1222.3154
Train Loss :46.9026
Valid Loss :1102.9030
Train Loss :44.9595
Valid Loss :1175.1011
Train Loss :41.8032
Valid Loss :1232.6340
Train Loss :43.5625
Valid Loss :1119.5381
Train Loss :31.8866
Valid Loss :1269.8796
Train Loss :37.4059
Valid Loss :1228.6127
Train Loss :33.9948
Valid Loss :1233.2611
Train Loss :29.3784
Valid Loss :1120.1033
Train Loss :28.4105
Valid Loss :1268.6351
```

[ ]:

[12]:
```
plt.plot(Loss_value,label='Train Loss')
plt.plot(val_Loss_value,label='Validation Loss')
plt.legend()
plt.title('Convergence Plots')
plt.xlabel('iterations')
plt.ylabel('Loss')
plt.grid()
```



[13]:
```
# get model
model.load_state_dict(torch.load('model.ckpt'))
```

[13]:
```
<All keys matched successfully>
```

[14]:
```
def test( model,test_loader, device):
    criterion = nn.MSELoss()
    model.load_state_dict(torch.load('model.ckpt'))
    with torch.no_grad():
        test_loss = 0
        pred_y_value,true_y_value = [],[]
        for X, Y in test_loader:
            X,Y= X.to(device),Y.to(device)
            model.to(device)
            predict = model(toTensor(X.reshape(-1,3,256,256)))
            loss = criterion(predict, Y)
            test_loss += loss.item() * X.size(0)

            true_y_value.extend(Y.cpu().detach().numpy().flatten())
            pred_y_value.extend(predict.cpu().detach().numpy().flatten())
```

```
    rmse_test = round(np.sqrt(mean_squared_error(true_y_value, pred_y_value)),5)
    pearson_test = round(pearsonr(true_y_value, pred_y_value)[0],5)
    spearman_test = round(spearmanr(true_y_value, pred_y_value)[0],5)
    r2_test = round(r2_score(true_y_value, pred_y_value),5)

    return true_y_value, pred_y_value, rmse_test, pearson_test, spearman_test,␣
↪r2_test
```

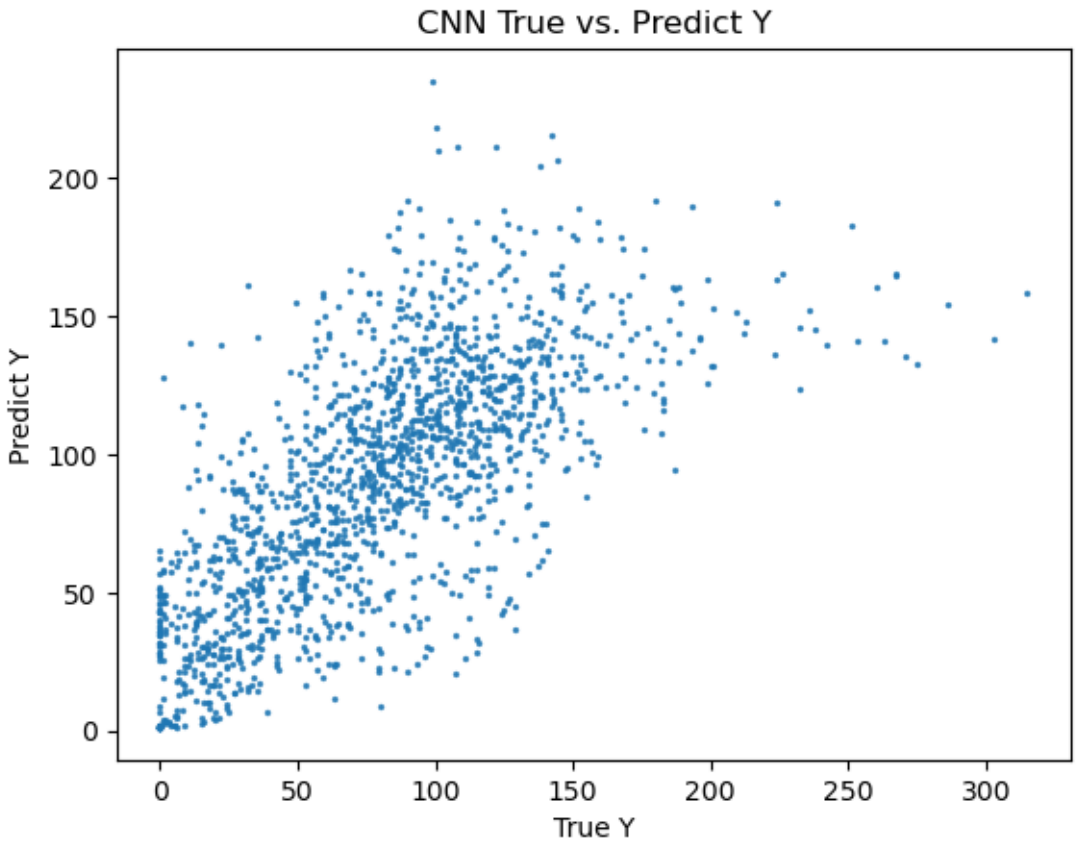
[15]:
```
true_y_value, pred_y_value, rmse_test, pearson_test, spearman_test, r2_test=␣
↪test(model,test_loader, device)
```

[14]:
```
print('Performance matrix of CNN: ')
print('RMSE:',rmse_test)
print('R^2:',r2_test)
print('Pearson Correlation Coefficient:',pearson_test)
print('Spearman Correlation Coefficient:',spearman_test)
```

```
Performance matrix of CNN:
RMSE: 37.98692
R^2: 0.40568
Pearson Correlation Coefficient: 0.70192
Spearman Correlation Coefficient: 0.72546
```

[15]:
```
plt.scatter(true_y_value,pred_y_value,s=2,alpha=0.8)
plt.title('CNN True vs. Predict Y')
plt.xlabel('True Y')
plt.ylabel('Predict Y')
```

[15]:
```
Text(0, 0.5, 'Predict Y')
```



3.0.2 b. Use a convolutional neural network (in Pytorch) to predict the counts of 6 types of cells simultaneously given the image patch as input as well as the total number of cells (7 outputs in total). You are free to choose any network structure as long as you can show that it gives good cross-validation performance. Report the results for the test fold for each cell type in the form of separate predicted-vs-actual count scatter plots (3 folds, 6 cell types and 1 as the total number of cells so 21 plots in total) using your optimal machine learning model and report your results in terms of RMSE, Pearson Correlation Coefficient, Spearman Correlation Coefficient and R2 score for each cell type and the total number of cells. [20 Marks]

4 Note the plots are at the end of output area

[15]:
```
New_Y=Y
```

[16]:
```
New_Y["total"] = New_Y[:].sum(axis =1)
New_Y.neutrophil
```

[16]:
```
0       0
1       0
2       0
3       0
4       0
       ..
4976    0
4977    0
4978    0
4979    0
4980    0
Name: neutrophil, Length: 4981, dtype: int64
```

[17]:
```
New_Y_fold1=New_Y[F==1]
New_Y_fold2=New_Y[F==2]
New_Y_fold3=New_Y[F==3]
```

[ ]:
```
#fold3 test is train
#fold2 val is test
#fold1 train is val
num_epochs=10
Type=['neutrophil','epithelial','lymphocyte','plasma','eosinophil','connective','total']
model_P_M=[['','rmse_test', 'pearson_test', 'spearman_test', 'r2_test']]

#train_loader,val_loader,test_loader=transform_data(X_train,Y_train,X_val,Y_val,X_test,Y_test)
fig, axs = plt.subplots(1, 7,figsize=(30,4))

for i in range(7):

    y_train3=np.array(New_Y_fold3[Type[i]])
    y_val1=np.array(New_Y_fold1[Type[i]])
    y_test2=np.array(New_Y_fold2[Type[i]])

    # process data
    ␣
 ↪train_loader,val_loader,test_loader=transform_data(X_test,y_train3,X_train,y_val1,X_val,y_t

    model1 = CellCountingCNN()
    model1.to(device)
    # Loss and optimizer
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model1.parameters(),␣
 ↪lr=learning_rate,weight_decay=5*(1e-4))

    # Train the model
    total_step = len(train_loader)
```

```
    val_loss_com=5000
    for epoch in range(num_epochs):
        loss_list=[]
        for  images, labels in tqdm(train_loader):
            images = images.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model1(toTensor(images.reshape(-1,3,256,256)))
            loss = criterion(outputs, labels)

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            loss_list.append(loss.item())

        train_loss = torch.tensor(loss_list).mean()

        print("Train Loss :{:.4f}".format(train_loss))
        val_loss=self_loss_function(model1,criterion,val_loader,device)
        print("Valid Loss :{:.4f}".format(val_loss))


        #Save the best parameters so far
        if (val_loss+train_loss)<val_loss_com:
          # Save the model checkpoint
            torch.save(model1.state_dict(), 'model1.ckpt')
            val_loss_com = val_loss+train_loss

    # get model
    model1.load_state_dict(torch.load('model1.ckpt'))
    true_y_value, pred_y_value, rmse_test, pearson_test, spearman_test,␣
 ↪r2_test= test(model1,test_loader, device)
    model_P_M.append([Type[i],rmse_test, pearson_test, spearman_test, r2_test])
    axs[i].scatter(true_y_value, pred_y_value,s=4,alpha=0.8)
    axs[i].set_xlabel('True Y')
    axs[i].set_ylabel('Predict Y')
    axs[i].set_title(Type[i]+':True vs Predict')
```

[ ]:
```
#fold3 test is val
#fold2 val is train
#fold1 train is test

num_epochs=10
Type=['neutrophil','epithelial','lymphocyte','plasma','eosinophil','connective','total']
model_P_M2=[['','rmse_test', 'pearson_test', 'spearman_test', 'r2_test']]
```

```
#train_loader,val_loader,test_loader=transform_data(X_train,Y_train,X_val,Y_val,X_test,Y_test)
fig, axs = plt.subplots(1, 7,figsize=(30,4))

for i in range(7):

    y_train3=np.array(New_Y_fold3[Type[i]])
    y_val1=np.array(New_Y_fold1[Type[i]])
    y_test2=np.array(New_Y_fold2[Type[i]])

    # process data
    
 ↪train_loader,val_loader,test_loader=transform_data(X_val,y_test2,X_test,y_train3,X_train,y_

    model2 = CellCountingCNN()
    model2.to(device)
    # Loss and optimizer
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model2.parameters(),
 ↪lr=learning_rate,weight_decay=5*(1e-4))

    # Train the model2
    total_step = len(train_loader)

    val_loss_com=5000
    for epoch in range(num_epochs):
        loss_list=[]
        for  images, labels in tqdm(train_loader):
            images = images.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model2(toTensor(images.reshape(-1,3,256,256)))
            loss = criterion(outputs, labels)

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            loss_list.append(loss.item())

        train_loss = torch.tensor(loss_list).mean()

        print("Train Loss :{:.4f}".format(train_loss))
        val_loss=self_loss_function(model2,criterion,val_loader,device)
        print("Valid Loss :{:.4f}".format(val_loss))
```

```
        #Save the best parameters so far
        if (val_loss+train_loss)<val_loss_com:
          # Save the model2 checkpoint
            torch.save(model2.state_dict(), 'model2.ckpt')
            val_loss_com = val_loss+train_loss

    # get model
    model2.load_state_dict(torch.load('model2.ckpt'))
    true_y_value, pred_y_value, rmse_test, pearson_test, spearman_test,␣
 ↪r2_test= test(model2,test_loader, device)

    model_P_M2.append([Type[i],rmse_test, pearson_test, spearman_test, r2_test])
    axs[i].scatter(true_y_value, pred_y_value,s=4,alpha=0.8)
    axs[i].set_xlabel('True Y')
    axs[i].set_ylabel('Predict Y')
    axs[i].set_title(Type[i]+':True vs Predict')
```

[ ]:

[ ]:
```
#fold1 train is train X_train,y_val1
#fold2 val is val     X_val,y_test2
#fold 3 test is test  X_test,y_train3


num_epochs=10
Type=['neutrophil','epithelial','lymphocyte','plasma','eosinophil','connective','total']

#train_loader,val_loader,test_loader=transform_data(X_train,Y_train,X_val,Y_val,X_test,Y_test)
fig, axs = plt.subplots(1, 7,figsize=(30,4))

model_P_M3=[['','rmse_test', 'pearson_test', 'spearman_test', 'r2_test']]
for i in range(7):

    y_train3=np.array(New_Y_fold3[Type[i]])
    y_val1=np.array(New_Y_fold1[Type[i]])
    y_test2=np.array(New_Y_fold2[Type[i]])

    # process data
  ␣
 ↪train_loader,val_loader,test_loader=transform_data(X_train,y_val1,X_val,y_test2,X_test,y_tr

    model3 = CellCountingCNN()
    model3.to(device)
    # Loss and optimizer
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model3.parameters(),␣
 ↪lr=learning_rate,weight_decay=5*(1e-4))
```

```
    # Train the model
    total_step = len(train_loader)

    val_loss_com=5000
    for epoch in range(num_epochs):
        loss_list=[]
        for  images, labels in tqdm(train_loader):
            images = images.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model3(toTensor(images.reshape(-1,3,256,256)))
            loss = criterion(outputs, labels)

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            loss_list.append(loss.item())

        train_loss = torch.tensor(loss_list).mean()

        print("Train Loss :{:.4f}".format(train_loss))
        val_loss=self_loss_function(model3,criterion,val_loader,device)
        print("Valid Loss :{:.4f}".format(val_loss))


        #Save the best parameters so far
        if (val_loss)<val_loss_com:
          # Save the model checkpoint
            torch.save(model3.state_dict(), 'model3.ckpt')
            val_loss_com = val_loss

    # get model
    model3.load_state_dict(torch.load('model3.ckpt'))
    true_y_value, pred_y_value, rmse_test, pearson_test, spearman_test,␣
 ↪r2_test= test(model3,test_loader, device)
    model_P_M3.append([Type[i],rmse_test, pearson_test, spearman_test, r2_test])
    axs[i].scatter(true_y_value, pred_y_value,s=4,alpha=0.8)
    axs[i].set_xlabel('True Y')
    axs[i].set_ylabel('Predict Y')
    axs[i].set_title(Type[i]+':True vs Predict')
```

[24]:
```
print('fold 3 as train dataset,fold 2 as test dataset and fold 1 as validation␣
 ↪dataset')
print(tabulate(model_P_M))
```

```
print('fold 2 as train dataset,fold 1 as test dataset and fold 3 as validation␣
 ↪dataset')
print(tabulate(model_P_M2))
print('fold 1 as train dataset,fold 3 as test dataset and fold 2 as validation␣
 ↪dataset')
print(tabulate(model_P_M3))
```

```
fold 3 as train dataset,fold 2 as test dataset and fold 1 as validation dataset
----------  ------------------  ------------  -------------  -----------
            rmse_test           pearson_test  spearman_test  r2_test
neutrophil  101.18834686279297  0.08328       0.0836         -1021.28971
epithelial  60.888671875         0.52295       0.5042         -2.34161
lymphocyte  85.3473129272461    0.4008        0.54387        -5.71069
plasma      97.23710632324219   0.33485       0.41472        -165.74172
eosinophil  101.4283218383789   0.21741       0.27468        -3748.50265
connective  84.12727355957031   0.29073       0.34544        -25.90799
total       34.051151275634766  0.7541        0.77062        0.55556
----------  ------------------  ------------  -------------  -----------
fold 2 as train dataset,fold 1 as test dataset and fold 3 as validation dataset
----------  ------------------  ------------  -------------  -----------
            rmse_test           pearson_test  spearman_test  r2_test
neutrophil  106.423828125       -0.01491      -0.00199       -952.43819
epithelial  62.00762176513672   0.55447       0.55868        -1.71552
lymphocyte  83.3658218383789    0.65435       0.66149        -7.40922
plasma      100.8328628540039   0.37235       0.50148        -150.05577
eosinophil  106.35932159423828  0.3103        0.30767        -6401.30848
connective  86.27129364013672   0.39761       0.42176        -24.40514
total       9.27322006225586    0.98682       0.98609        0.96625
----------  ------------------  ------------  -------------  -----------
fold 1 as train dataset,fold 3 as test dataset and fold 2 as validation dataset
----------  ------------------  ------------  -------------  -----------
            rmse_test           pearson_test  spearman_test  r2_test
neutrophil  103.21112060546875  0.12931       0.17862        -4018.35369
epithelial  69.00921630859375   0.45171       0.43824        -3.09771
lymphocyte  88.42082977294922   0.3735        0.58704        -7.95444
plasma      99.08627319335938   0.18966       0.31506        -131.03655
eosinophil  102.92286682128906  0.26063       0.30308        -4185.98924
connective  86.38395690917969   0.27566       0.29348        -21.86095
total       39.928890228271484  0.68055       0.7105         0.34336
----------  ------------------  ------------  -------------  -----------
```

[ ]: