Image Features Code-book

Aneesh Sathe

January 10, 2017

Contents

1 Reading images and finding nuclei

1.1 Description

This is the driver code. This part of the code is run in parallel by an outer script. `an_img_prop` is the main function. The function takes in the image array or a file list and various strings used to specify the name of the saved image. In case of file list the list is read into memory.

Currently all the images are saved in `.hdf5` format using a separat(setq dotspacemacs-additional-packages '(org-ref))e script. Here each((require 'org-ref)setq dotspacemacs-additional-packages '(org-ref)) image can be tagged with its appropriate class. e.g. NIH3T3 for control NIH3T3 cells and $\text{NIH3T3}_{\text{Load}}$ for NIH3T3 with load. The name of the hdf5 file itself specifies the experiment date, and type of experiment.

The image class is used as a prefix and the experiment date is used as a suffix for the name of the saved imaged. As the code is run in parallel the processor number and image index are used to generate unique names.

For every image a rough intensity threshold is applied. From the generated mask and the original image regions props function is used to determine area and intensity properties. Only those regions with area greater then 1500 pixels and intensity greater than 0.02 are regarded as nuclei.

Due to high-throughput imaging, there are some images that have no nuclei, these images are rejected if the standard deviation of the intensity is less than 20.

After the nuclei are identified the cropped region of the nuclei are sent to various functions for feature extraction. The Zernike moments perform better if there is no background, so cropped images multiplied by their masks are used.

The cropped nuclei are padded to be of size 256x256 pixels and saved.

1.2 Code

```
from skimage.filters import threshold_otsu
from skimage.morphology import closing, square
from skimage.morphology import disk
from skimage.morphology import opening
from skimage.io import imread_collection
from skimage.io import imsave
from skimage.measure import regionprops
import numpy as np
```

```
from skimage import img_as_uint
from skimage.util import pad
from scipy import ndimage as ndi
from skimage.morphology import watershed
from skimage.feature import peak_local_max
import os
import pandas as pd
from an_img_baseprops import an_base_props, an_stat_props
from an_glcm_props import glcm_props
from an_glrlm_props import glrlm_props
from an_spectral_props import an_get_spec_feats
from an_spatial_props import an_laws_feats
from an_spatial_diversity_props import an_spa_div
from an_lbp_tas_zernike import an_lbp_props, an_tas_props, an_zernike_props


def an_get_img_prop(in_array=None, img_idx=None, pool_num=0, file_list=None, lbl_name=N

    if file_list is not None:
        img_coll = imread_collection(file_list)
    elif in_array is not None:
        img_coll = in_array
    else:
        print('Invalid input array or file list')
        return None

    feat_frame = pd.DataFrame([])
    row_count = 0
    im_count = 0

    for im_num, image in enumerate(img_coll):
        if np.std(image) > 20:  # For empty images
            if lbl_name is None:
                sav_lbl = 'No_Label'
            else:
                sav_lbl = lbl_name[img_idx[im_num]]
        #        print(sav_lbl)
            label_image = an_thresh_lbl_img(image)

            print('Image ' + str(im_num) + ' of ' + str(len(img_coll))+' Proc num: '+st
            for region_prop in regionprops(label_image, image):

                # skip small images
                if region_prop.area < 1500 or region_prop.max_intensity < 0.02:
                    continue
                minr, minc, maxr, maxc = region_prop.bbox
                crop_im = image[minr:maxr, minc:maxc].copy()
                bg_zero_im = crop_im * region_prop.image
                feat_frame.loc[row_count, 'CellType'] = sav_lbl
                feat_frame = an_base_props(feat_frame, row_count, region_prop, crop_im
                feat_frame = an_stat_props(feat_frame, row_count, region_prop, crop_im
                feat_frame = glcm_props(feat_frame, row_count, crop_im)
                feat_frame = glrlm_props(feat_frame, row_count, crop_im)
                feat_frame = an_get_spec_feats(feat_frame, row_count, crop_im)
                feat_frame = an_laws_feats(feat_frame, row_count, crop_im)
                feat_frame = an_spa_div(feat_frame, row_count, crop_im)
                feat_frame = an_lbp_props(feat_frame, row_count, bg_zero_im)
                feat_frame = an_tas_props(feat_frame, row_count, crop_im)
                feat_frame = an_zernike_props(feat_frame, row_count, bg_zero_im)
                row_count += 1

                im_count += 1
                if img_write_path is not None:
                    crop_im = an_pad_im(region_prop.image*crop_im)
                    an_sav_im(img_write_path, crop_im, sav_lbl, pool_num, im_count)

        feat_frame.fillna(np.finfo(float).eps, inplace=True)
    return feat_frame
```

2 Base Properties (BP_)

2.1 Properties extracted:

Properties are extracted using the regionprops function from the Scikit-Image package: `skimage.measure.regionprops`

Properties	Code
Area	'Area'
Eccentricity	'Eccentricity'
Euler Number	'Euler$_{\text{Num}}$'
Equivalent diameter	'Equi$_{\text{Diam}}$'
Extent	'Extent'
Major axis length	'Maj$_{\text{axlen}}$'
Minor axis length	'Min$_{\text{axlen}}$'
Maximum intensity	'Max$_{\text{int}}$'
Mean intensity	'Mean$_{\text{int}}$'
Minimum intensity	'Min$_{\text{int}}$'
Orientation	'Orientation'
Perimeter	'Perimeter'
Solidity	'Solidity'
Roundness	'Roundness'
Kurtosis	'Kurtosis'
Skewness	'Skewness'
Image moments	'moments'
Weighted image moments	'wt$_{\text{moments}}$'
Weighted central moments	'wt$_{\text{cenmoments}}$'
Weighted normalized moments	'wt$_{\text{normmoments}}$'
Hu's moments (14)	'hu$_{\text{moments}}$'
Hu's weighted moments (14)	'hu$_{\text{wtmoments}}$'

2.2 Code

```
def an_base_props(feat_frame, row_count, region_prop, crop_im):
    out_props = [region_prop.area, region_prop.eccentricity,
                 region_prop.euler_number, region_prop.equivalent_diameter,
                 region_prop.extent, region_prop.major_axis_length,
                 region_prop.minor_axis_length, region_prop.max_intensity,
                 region_prop.mean_intensity, region_prop.min_intensity,
                 region_prop.orientation, region_prop.perimeter,
                 region_prop.solidity, mahotas.features.roundness(region_prop.image),
                 scipy.stats.kurtosis(crop_im.flatten()), scipy.stats.skew(crop_im.fla
                 ]
    out_lbl = ['Area', 'Eccentricity',
               'Euler_Num', 'Equi_Diam',
               'Extent', 'Maj_ax_len',
               'Min_ax_len', 'Max_int',
                'Mean_int', 'Min_int',
                'Orientation', 'Perimeter',
                'Solidity', 'Roundness',
                'Kurtosis', 'Skewness']

    [(out_props.append(mom), out_lbl.append('moments_' + str(i) + '_')) for i, mom in
     enumerate(np.array(region_prop.moments).flatten())]
    [(out_props.append(mom), out_lbl.append('wt_moments_' + str(i) + '_')) for i, mom i
     enumerate(np.array(region_prop.weighted_moments).flatten())]
    [(out_props.append(mom), out_lbl.append('wt_cen_moments_' + str(i) + '_')) for i, m
     enumerate(np.array(region_prop.weighted_moments_central).flatten())]
    [(out_props.append(mom), out_lbl.append('wt_norm_moments_' + str(i) + '_')) for i,
     enumerate(np.array(region_prop.weighted_moments_normalized).flatten())]
    [(out_props.append(mom), out_lbl.append('hu_moments_' + str(i) + '_')) for i, mom i
     enumerate(np.array(region_prop.moments_hu).flatten())]
    [(out_props.append(mom), out_lbl.append('hu_wt_moments_' + str(i) + '_')) for i, mo
     enumerate(np.array(region_prop.weighted_moments_hu).flatten())]

    out_lbl = ['BP_' + lbl + '_' for lbl in out_lbl]

    out_props = np.array(out_props)
    out_props[np.isnan(out_props)] = np.finfo(float).eps

    # feat_frame = an_df_filler(feat_frame, row_count, out_props, ['BaseProps'])
    feat_frame = an_df_filler(feat_frame, row_count, out_props, out_lbl)

    return feat_frame
```

3 Statistical Texture Features (ST)

3.1 First order statistics (STFOS_)

Ref: Virmani, J., Kumar, V., Kalra, N., Khandelwal, N.: A rapid approach for prediction of liver cirrhosis based on first order statistics. In: Proceedings of IEEE International Conference on Multimedia, Signal Processing and Communication Technologies, IMPACT-2011, pp. 212– 215 (2011)

Properties	Code
Average Gray Level	'Mean_{int}'
Smoothness	'Smoothness'
Standard Deviation	'StDev'
Entropy	'Entropy'
Uniformity	'Uniformity'
Skewness	'Skewness'

- Smoothness: Smoothness describes weather the texture of the image is smooth or rough. The texture is smooth if the image contains constant gray level intensity values and is considered as rough if there are rapid variations in the intensity levels.
- Entropy: Entropy gives the measure of randomness in the gray level intensity values with in an image. `img_ent = sum(bin_prob*np.log2(bin_prob))`
- Skewness (Third moment): Skewness indicates weather the histogram of intensity levels in the image is symmetric about the mean
- Uniformity: Uniformity gives the maximum value when all the gray levels in the image are equal to indicate that the texture is maximally uniform. `sum(np.square(bin_prob))`

3.1.1 Code:

```
def an_stat_props(feat_frame, row_count, region_prop, crop_im):
    pro_bin_prob, bin_cen = histogram(crop_im[region_prop.image])
    bin_prob = np.float16(pro_bin_prob) / sum(pro_bin_prob)

    out_props = [region_prop.mean_intensity,  # mean intensity
                 1.0 - (1.0 / (1.0 + np.square(np.std(region_prop.intensity_image)))),
                 np.std(crop_im[region_prop.image]),  # std
                 sum(bin_prob * np.log2(bin_prob)),  # entropy
                 sum(np.square(bin_prob)),  # Uniformity
                 scipy.stats.skew(np.ndarray.flatten(crop_im[region_prop.image])),  # s
                 ]
    out_lbl = ['Mean_int', 'Smoothness', 'StDev', 'Entropy', 'Uniformity', 'Skewness']
    out_lbl = ['STFOS_' + lbl + '_' for lbl in out_lbl]

    feat_frame = an_df_filler(feat_frame, row_count, out_props, out_lbl)
    # feat_frame = an_df_filler(feat_frame, row_count, out_props, ['STFOS'])
    return feat_frame
```

3.2 Second order statistics : GLCM Features (STSOS)

REF: `http://www.fp.ucalgary.ca/mhallbey/the_glcm.htm`

Other References:

1. R. M. Haralick, K. Shanmugam, and I. Dinstein, Textural Features of

Image Classification, IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-3, no. 6, Nov. 1973

1. L. Soh and C. Tsatsoulis, Texture Analysis of SAR Sea Ice Imagery

Using Gray Level Co-Occurrence Matrices, IEEE Transactions on Geoscience and Remote Sensing, vol. 37, no. 2, March 1999.

1. D A. Clausi, An analysis of co-occurrence texture statistics as a

function of grey level quantization, Can. J. Remote Sensing, vol. 28, no. 1, pp. 45-62, 2002

1. `http://murphylab.web.cmu.edu/publications/boland/boland_node26.html`

GLCM considers the spatial relationship of pixels and characterizes the texture of an image by calculating how often pairs of pixel with specific values and spatial relationships occur in an image. Once the GLCM is constructed, statistical measures are extracted from the matrix.

The $P_{i,j}$ used below is defined here: `http://www.fp.ucalgary.ca/mhallbey/GLCM_as_probability.htm` Refers to individual numbers in the GLCM

When calculating GLCM with skimage the symmetric and normed arguments are set to 'True' to obtain the symmetric probability matrix.

GLCM is calculated at combinations of 4 distances and 4 angles:

Distances	1, 2, 4, 8, 10
Angles	0, 45, 90, 135

Features

Properties	Code
Contrast	'Con'
Dissimilarity	'Dsm'
Homogeneity	'Hom'
angular second moment	'ASM'
Energy	'Enr'
Correlation	'Cor'
Entropy	'Ent'
sum average	'muSum'
variance	'Var'
sum variance	'SumVariance'
difference variance	'DiffVariance'
GLCM means	'muSum'
sum entropy	'SumEntropy'
difference entropy	'DiffEntropy'
information measure of correlation-1	'IMC1'
information measure of correlation-2	'IMC2'
GLCM Single Value Decomposition	'SVD'

$\mathrm{P_{xplusy}}$ and $\mathrm{P_{xminusy}}$ are defined as below

$$p_{x+y}(k) = \sum_{i,j=0}^{N-1} P_{i,j}(i,j) where k = i + j = 2, 3, ..., 2N$$

$$p_{x-y}(k) = \sum_{i,j=0}^{N-1} P_{i,j}(i,j) where k = |i - j| = 0, 1, ..., N - 1$$

3.2.1 Feature Details*: Some of these such as variance as calculated for every GLCM. Others are calculated across GLCMs

- Contrast: Also called "sum of squares variance"

$$c = \sum_{i,j=0}^{N-1} P_{i,j}(i-j)^2$$

- Dissimilarity:

$$d = \sum_{i,j=0}^{N-1} P_{i,j}|i-j|$$

- Homogeneity (inverse difference moment): Homogeneity weights values by the inverse of the Contrast weight, with weights decreasing exponentially away from the diagonal:

$$homogen = \sum_{i,j=0}^{N-1} \frac{P_{i,j}}{1+(i-j)^2}$$

- Angular Second Moment (ASM): ASM and Energy use each Pij as a weight for itself. High values of ASM or Energy occur when the window is very orderly.

$$ASM = \sum_{i,j=0}^{N-1} P_{i,j}^2$$

- Energy: $Energy = \sqrt{ASM}$
- Correlation:
- Entropy: $Entropy = \sum_{i,j=0}^{N-1} P_{i,j}(-lnP_{i,j})$ where: $ln(\inf) = 0$
- GLCM Means: The left hand equation calculates the mean based on the reference pixels, i. It is also possible to calculate the mean using the neighbour pixels, j, as in the right hand equation. For the symmetrical GLCM, where each pixel in the window is counted once as a reference and once as a neighbour, the two values are identical. $\mu_i = \sum_{i,j=0}^{N-1} i(P_{i,j})\mu_j = \sum_{i,j=0}^{N-1} j(P_{i,j})$

Currently, we are using symmetric glcm so calculate only one.

- Sum Average:

$$sumAvg = \sum_{i=2}^{2N_g} ip_{x+y}(i)$$

- Variance:

$$Var = \sum_{i,j=0}^{N-1} (i-\mu)^2 P_{i,j}$$

- Sum Variance:

$$SumVariance = \sum_{i=2}^{2N_g} (i - sumAvg)^2 p_{x+y}(i))$$

- Difference Variance:

$$DiffVariance = var(p_{x-y})$$

- Sum Entropy:

$$SumEntropy = \sum_{i=2}^{2N_g} p_{x+y}(i) ln(p_{x+y}(i))$$

- Difference Entropy:

$$DiffEntropy = \sum_{i=0}^{N_{g-1}} p_{x-y}(i) ln(p_{x-y}(i))$$

- Information measures of correlation (1 & 2): reference: `https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/22354/versions/5/previews/GLCM_Features4.m/index.html`

out.inf1h(k) = (hxy(k) - hxy1(k)) / (max([hx(k),hy(k)]));

- GLCM SVD: The Single Value Decompositions are returned the output dimension is dist*ang*$no_{ofval2return}$

where dist and ang are the number of distances and angles given to glcm, $no_{ofval2return}$ is number of top svd outputs to be returned. The output can be grouped by the $no_{ofval2return}$ and will be of the order [distance[angle[value]]]
SVD: `https://www.ling.ohio-state.edu/~kbaker/pubs/Singular_Value_Decomposition_Tutorial.pdf`

3.2.2 Code:

```
import numpy as np
import scipy
from skimage import img_as_ubyte
from skimage.feature import greycomatrix, greycoprops

from helper_func import an_df_filler


def glcm_props(feat_frame, row_num, crop_im):
    in_dist = [1, 2, 4, 8, 10]
```

```
in_angles = [0, np.pi / 4, np.pi / 2, 3 * np.pi / 4]

glcm = greycomatrix(img_as_ubyte(crop_im), in_dist, in_angles, symmetric=True, norm
# Replace 0s with small value
glcm += np.finfo(float).eps

out_props = [an_glcm_cont(glcm),  # Contrasts, their means and ranges
             an_glcm_dissim(glcm),  # Dissimilarity
             an_glcm_homogen(glcm),  # Homogeneity
             an_glcm_asm(glcm),  # Angular Second Moment
             an_glcm_ener(glcm),  # Energy
             an_glcm_corr(glcm),  # Correlation
             an_iter_func_over_glcm(an_glcm_entropy, glcm),  # Entropy
             ]

no_of_val2return = 15  # for SVD

out_props = np.ravel(out_props)
out_props = np.concatenate((out_props,
                            glcm_more_feats(glcm), # ['sumAvg','SumVariance','DiffV
                            an_glcm_svd(glcm, no_of_val2return=no_of_val2return)  #
                            ))
# print(out_props.shape)
u_ang = np.degrees(in_angles).astype(int)

b_lbl = ['Con', 'Dsm', 'Hom', 'ASM', 'Enr', 'Cor', 'Ent', ]
out_lbl = np.ravel([glcm_nm(a, u_ang, in_dist, lbl_type='Base') for a in b_lbl])

b_lbl = ['muSum', 'Var', 'sumAvg', 'SumVariance', 'DiffVariance', 'SumEntropy', 'Di
out_lbl = np.append(out_lbl, np.ravel(glcm_nm(b_lbl, u_ang, in_dist, lbl_type='More

out_lbl = np.append(out_lbl,
                    np.ravel(glcm_nm(b_lbl, u_ang, in_dist, lbl_type='SVD', no_of_v
# print(feat_frame.shape)
# print(out_props.shape)
# print(out_lbl.shape)

#     feat_frame = an_df_filler(feat_frame, row_num, out_props, ['STSOS'])
feat_frame = an_df_filler(feat_frame, row_num, out_props, out_lbl)  # ['STSOS'])
return feat_frame
```

```
# HELPER FUNCTIONS


def an_glc_mean(in_prp):
    """
    Calculates mean of incoming property along axis 1
    :param in_prp:
    :return:
    """
    return np.sum(in_prp, axis=1) / in_prp.shape[1]


def an_glc_range(in_prp):
    """
    Calculates range of incoming property along axis 1
    :param in_prp:
    :return:
    """
    return np.max(in_prp, axis=1) - np.min(in_prp, axis=1)


def an_join_prop_mean_range(in_prop):
    """
    Joins incoming features and their mean and range values by row. Then returns as si
    :param in_prop:
    :return:
    """
    in_mean = np.sum(in_prop, axis=1) / in_prop.shape[1]
    # in_mean = an_glc_mean(in_prop)
    in_range = np.max(in_prop, axis=1) - np.min(in_prop, axis=1)
    # in_range = an_glc_range(in_prop)
    # print(in_mean)
    # print(in_range)
    return np.ravel([np.append(in_prop[i], [in_mean[i], in_range[i]]) for i, _ in enume


def an_iter_func_over_glcm(in_func, in_glcm):
    """
    Applies incoming function in_func to incoming glcm, in_glcm
    :param in_func:
    :param in_glcm:
    :return:
    """
    prop = [in_func(in_glcm[:, :, i, j]) for i in range(in_glcm.shape[-2]) for j in ran
    prop = np.array(prop).reshape((in_glcm.shape[-2], in_glcm.shape[-1]))
    # print prop
    return an_join_prop_mean_range(prop)


def an_glcm_base_vals(in_glcm):
    f = np.array([in_glcm[i, j] for i in range(in_glcm.shape[-2]) for j in range(in_glc
                  if i+j in range(2, 2*in_glcm.shape[0])])
    p_xplusy = f[range(2, 2*in_glcm.shape[0])]
    p_xminusy = f[range(0, in_glcm.shape[0]-1)]
    # print(p_xplusy.shape)
    # print(p_xminusy.shape)
    return p_xplusy, p_xminusy

# FEATURE FUNCTIONS


def an_glcm_cont(glcm):
    """
    Calculates Contrast, mean and range for each glcm matrix
    :param glcm:
    :return: row in form [contrast 1.1, contrast 1.2,...mean1, range1, contrast 2.1..]
    """
    cont = greycoprops(glcm, 'contrast')  #
    # print(cont)
    return an_join_prop_mean_range(cont)


def an_glcm_dissim(glcm):
    """
    Calculates Dissimilarity of incoming glcm. Returns dissimilarity, mean, range like
    :param glcm:
    :return:
    """
    dissim = greycoprops(glcm, 'dissimilarity')
    # print(dissim)
    return an_join_prop_mean_range(dissim)


def an_glcm_homogen(glcm):
    """
    Calculates homogeneity of incoming glcm
    :param glcm:
    :return:
    """
    homogen = greycoprops(glcm, 'homogeneity')
    # print(homogen)

    return an_join_prop_mean_range(homogen)


def an_glcm_asm(glcm):
    """
    Calculates Angular Second Moment
    :param glcm:
    :return:
    """
    asm = greycoprops(glcm, 'ASM')
    return an_join_prop_mean_range(asm)


def an_glcm_ener(glcm):
    """
    Calculates Energy of incoming GLCMs
    :param glcm:
    :return:
    """
    ener = greycoprops(glcm, 'energy')
    # print(ener)
    return an_join_prop_mean_range(ener)


def an_glcm_corr(glcm):
    """
    Calculates Correlation of incoming GLCMs
    :param glcm:
    :return:
    """
    corr = greycoprops(glcm, 'correlation')
    # print(corr)
    return an_join_prop_mean_range(corr)


def an_glcm_entropy(in_glcm):
    pro_ent = -np.log(in_glcm)
    pro_ent[np.isinf(pro_ent) | np.isneginf(pro_ent)] = np.finfo(float).eps
    out_ent = np.sum(in_glcm*pro_ent)
    return out_ent


def glcm_more_feats(glcm):
    """
    Calculates features with shared variables.
    :param glcm:
    :return:
    """
    out_l = []

    for count1 in range(glcm.shape[-2]):
        for count2 in range(glcm.shape[-1]):

            in_glcm = glcm[:, :, count1, count2]
            p_xplusy, p_xminusy = an_glcm_base_vals(in_glcm)

            mu_i = [in_glcm[i, :] * i for i in range(0, in_glcm.shape[0])]
            i_minus_usq = ((range(in_glcm.shape[0]) - mu_i[0]) ** 2)
            an_var = np.sum([i_minus_usq[i] * in_glcm[i, :] for i in range(in_glcm.sha

            sumAvg = np.sum(range(2, 2*in_glcm.shape[0])*p_xplusy)
            SumVariance = np.sum((np.array(range(2,2*in_glcm.shape[0])-sumAvg)**2)*p_x
            DiffVariance = np.var(p_xminusy)
            SumEntropy = -np.sum(p_xplusy*np.log(p_xplusy))
            DiffEntropy = np.sum(-p_xminusy*np.log(p_xminusy))
            px_i = np.sum(in_glcm, axis=1)
            py_j = np.sum(in_glcm, axis=0)
            HX = scipy.stats.entropy(px_i)
            HY = scipy.stats.entropy(py_j)
            HXY  = -np.sum([in_glcm[i, j]*np.log(in_glcm[i, j]) for i in range(in_glcm
            HXY1 = -np.sum([in_glcm[i, j]*np.log(px_i[i]*py_j[j]) for i in range(in_glc
            HXY2 = -np.sum([px_i[i]*py_j[j]*np.log(px_i[i]*py_j[j]) for i in range(in_g
            IMC1 = HXY-HXY1/np.max([HX, HY])
            IMC2 = np.sqrt(1-np.exp(-2*(HXY2-HXY)))
#             print IMC2

            out_l.append([np.sum(mu_i),
                          an_var,
                          sumAvg,
                          SumVariance,
                          DiffVariance,
                          SumEntropy,
                          DiffEntropy,
                          IMC1,
                          IMC2
                          ])

    return np.ravel(out_l)


def an_glcm_svd(in_glcm, no_of_val2return=15):
    """
    Calculates the single value decompositions of the incoming glcm and returns the dia
    the most informative values(15 by default).
    :param in_glcm:
    :param no_of_val2return:
    :return:
    """
    pro_out_svd = []
    for i in range(0, in_glcm.shape[3]):
        u, s, v = np.linalg.svd(in_glcm[:, :, :, i])
        s_sorted = -np.sort(-s, axis=0)
        pro_out_svd.append(np.transpose(s_sorted[0:no_of_val2return]))
        # pyplot.plot(s_sorted[0:no_of_val2return])
    out_svd = np.ravel(pro_out_svd)

    return out_svd


def glcm_nm(in_nm,in_angles, in_dist, lbl_type, no_of_val2return=0):

    a_lbl = [['GLCM_t_'+str(a) +'_d_'+str(d) for a in in_angles] for d in in_dist ]

    if (lbl_type=='Base'):
        b_lbl = [np.append(a, ['GLCM_d_'+str(in_dist[i])+'_avg','GLCM_d_'+str(in_dist[
        return [b+'_'+in_nm+'_' for b in np.ravel(b_lbl)]
    elif (lbl_type=='More'):
        return [b+'_'+c+'_' for b in np.ravel(a_lbl) for c in in_nm]
    elif (lbl_type=='SVD'):
        s_lbl = ['SVD'+str(s) for s in range(1, no_of_val2return+1)]
        return [b+'_'+c+'_' for b in np.ravel(a_lbl) for c in s_lbl]
```

3.3 Higher order statistics: GLRLM features (STHOS)

GLRLM features are calculated at a distance of 2 pixels and angles 0, 45, 90, 135

Main paper: Xu, D.-H., Kurani, A. S., Furst, J. D., & Raicu, D. S. (2004). Run-Length Encoding for Volumetric Texture. International Conference on Visualization, Imaging and Image Processing (VIIP), 452–458.

Paper: `http://ac.els-cdn.com/S0146664X75800086/1-s2.0-S0146664X75800086-main.pdf?_tid=e2522c20-fc7c-11e5-979a-00000aab0f26&acdnat=1460005080_1ab137efee9f26b0842f2316c7df42ab`

Properties	Code
long run emphasis	'LRE'
short run emphasis	'SRE'
high gray level run emphasis	'HGLRE'
low gray level run emphasis	'LGLRE'
short run high gray level emphasis	'SRHGLE'
short run low gray level emphasis	'SRLGLE'
long run high gray level emphasis	'LRHGLE'
long run low gray level emphasis	'LRLGLE'
gray level non-uniformity	'GLN'
run length non-uniformity	'RLN'
run percentage	'RP'

Output features are labelled as $\mathrm{STHOS}_{\mathrm{r2a0LRE}}$ which in this case means radius of 2, angle of 0 and LRE.

REF; `https://github.com/acil-bwh/SlicerCIP/blob/7f93e53b9befd39d1e7231660b637c3d6108` `Scripted/CIP_LesionModel/FeatureExtractionLib/TextureGLRL.py`

*FOR SRE,LRE,GLN,RLN & RP: *

Galloway, M. M. (1975). Texture analysis using gray level run lengths. Computer Graphics and Image Processing, 4(2), 172–179. `http://doi.org/10.1016/S0146-664X(75)`80008-6

FOR HGLRE, LGLRE, SRHGLE, SRLGLE, LRHGLE, & LRLGLE :

Chu, A., Sehgal, C. M., & Greenleaf, J. F. (1990). Use of gray value distribution of run lengths for texture analysis. Pattern Recognition Letters, 11(6), 415–419. `http://doi.org/10.1016/0167-8655(90)`90112-F

3.3.1 Feature Details:

- short run emphasis (SRE): This function divides each run length value by the length of the run squared.

This tends to emphasize short runs. This should emphasize long runs. The denominator is a normalizing factor, as above.

$$SRE = \sum_{i=1}^{N_g}\sum_{j=1}^{N_r}\frac{p(i,j)}{j^2}\Bigg/\sum_{i=1}^{N_g}\sum_{j=1}^{N_r}p(i,j) = \sum_{j=1}^{N_r}\frac{r(j)/s}{j^2}$$

- long run emphasis (LRE): his function multiplies each run length value by the length of the run squared. The denominator is the total number of runs in the picture and serves as a normalizing factor.

$$LRE = \sum_{i=1}^{N_g}\sum_{j=1}^{N_r}j^2p(i,j)\Bigg/\sum_{i=1}^{N_g}\sum_{j=1}^{N_r}p(i,j) = \sum_{j=1}^{N_r}r(j)j^2/s$$

- gray level non-uniformity (GLN) : This function squares the number of run lengths for each gray level. The sum

of the squares is then divided by the normalizing factor of the total number of runs in the picture This should measure the gray level non-uniformity of the picture. When runs are equally distributed throughout the gray levels, the function takes on its lowest values. High run length values contribute most to the function.

$$GLN = \sum_{i=1}^{N_g}\left(\sum_{j=1}^{N_r} p(i,j)\right)^2 \Big/ \sum_{i=1}^{N_g}\sum_{j=1}^{N_r} p(i,j)$$

- run length non-uniformity (RLN) : This function squares the number of runs for each length. The sum of the squares is then divided by the normalizing factor. This function measures the non-uniformity of the run lengths. If the runs are equally distributed throughout the lengths, the function will have a low value. Large run counts contribute most to the function.

$$RLN = \sum_{j=1}^{N_g}\left(\sum_{i=1}^{N_r} p(i,j)\right)^2 \Big/ \sum_{i=1}^{N_g}\sum_{j=1}^{N_r} p(i,j)$$

- Run percentage (RP): This function is a ratio of the total number of runs to the total number of possible runs if all runs had a length of one. It should have its lowest value for pictures with the most linear structure.

Aneesh: the original paper makes this set for distance of 1 and is only divided by the size of the image. To account for different radii the img.size has to be divided by pix_{dist} (radii)

$$RP = \sum_{i=1}^{N_g}\sum_{j=1}^{N_r} p(i,j)/P$$

- Low gray level run emphasis (LGLRE): Measures the distribution of low gray level values. The LGRE is expected large for the image with low gray level values.

$$LGLRE = \sum_{i=1}^{N_g}\sum_{j=1}^{N_r} \frac{g(i)/s}{i^2}$$

- High gray level run emphasis (HGLRE): Measures the distribution of high gray level values. The HGRE is expected large for the image with high gray level values.

$$HGLRE = \sum_{i=1}^{N_g}\sum_{j=1}^{N_r} i^2 g(i)/s$$

- short run low gray level emphasis (SRLGLE): Measures the joint distribution of short runs and low gray level values. The SRLGE is expected large for the image with many short runs and lower gray level values

$$SRLGLE = \frac{1}{s} \sum_{i=1}^{N_g} \sum_{j=1}^{N_r} \frac{p(i,j)}{i^2 j^2}$$

- short run high gray level emphasis (SRHGLE): Measures the joint distribution of short runs and high gray level values. The SRHGE is expected large for the image with many short runs and high gray level values

$$SRHGLE = \frac{1}{s} \sum_{i=1}^{N_g} \sum_{j=1}^{N_r} \frac{p(i,j) i^2}{j^2}$$

- long run low gray level emphasis (LRLGLE): Measures the joint distribution of long runs and low gray level values. The LRLGE is expected large for the image with many long runs and low gray level values

$$LRLGLE = \frac{1}{s} \sum_{i=1}^{N_g} \sum_{j=1}^{N_r} \frac{p(i,j) j^2}{i^2}$$

- long run high gray level emphasis (LRHGLE):

$$LRHGLE = \frac{1}{s} \sum_{i=1}^{N_g} \sum_{j=1}^{N_r} p(i,j) i^2 j^2$$

3.3.2 Code:

```
from math import *
import numpy as np
from skimage import img_as_ubyte
from helper_func import an_df_filler


def glrlm_props(feat_frame, row_num, crop_im, pix_dist=[2], pix_angle=[0, 45, 90, 135]
    """
    Calculates Gray Level Run Length features. Default distance is 2 and angles 0,45,9
    :param feat_frame:
    :param crop_im:
    :param pix_dist:
    :param pix_angle:
    :return:
    """

    # pix_dist = [1, 2, 4, 8]
    # pix_angle = [0, 45, 90, 135]

    out_glrlm = [get_glrlm_props(crop_im, dist, ang) for dist in pix_dist for ang in pi
    out_glrlm = np.ravel(out_glrlm)

    out_l = ['SRE', 'LRE', 'GLN', 'RLN', 'RP','LGLRE','HGLRE', 'SRLGLE', 'SRHGLE', 'LRI
    out_lbl = ['STHOS_r_'+str(dist)+'_a_'+str(ang)+'_'+l+'_' for dist in pix_dist for a
    # print(out_lbl)
    # print(np.ravel(out_glrlm).shape)

    feat_frame = an_df_filler(feat_frame, row_num, out_glrlm, out_lbl)


    return feat_frame

# Helper Functions


def an_get_pts_at_dist(in_row, in_col, in_radius, in_angle):
    out_row = sin(radians(in_angle))*in_radius
    out_col = cos(radians(in_angle))*in_radius

    if out_row < 0:
        out_row = floor(out_row)
    else:
        out_row = ceil(out_row)
    if out_col < 0:
        out_col = floor(out_col)
    else:
        out_col = ceil(out_col)

    out_row = int(in_row+out_row)
    out_col = int(in_col+out_col)
    return out_row, out_col


# Prop Functions


def get_glrlm_props(crop_im, pix_dist, pix_angle):
    img = img_as_ubyte(crop_im)
    # print(img.shape)
    max_int = np.max(img)
    min_int = np.min(img)
    gl = (max_int-min_int)+1

    # imshow(skimage.exposure.equalize_hist(img))
    mc = 0
    # pix_dist = 2
    # pix_angle = 45
    count = 1
    c = 0
    col = 0
    maxcount = np.zeros((img.shape[0] * img.shape[1]))
    grl = np.zeros((max_int, max([len(np.diagonal(img)), img.shape[0], img.shape[1]]))

    for row_count in range(0, img.shape[0] - pix_dist):

        for col_count in range(0, img.shape[1] - pix_dist):

            mc += 1
            ref_pix = img[row_count, col_count]
            test_pix = img[an_get_pts_at_dist(row_count - 1, col_count - 1, pix_dist, 

            if ref_pix == test_pix & ref_pix != 0:
                count += 1
                c = count
                col = count
                maxcount[mc] = count
            else:
                grl[ref_pix - 1, c] += 1;
                col = 1
                count = 1
                c = 0

        grl[test_pix - 1, col - 1] += 1
        count = 1
        c = 1

        # print(grl)
    # print(grl.shape)
    np.max(maxcount)

    g_row = np.sum(grl, axis=0)
    r_col = np.sum(grl, axis=1)
    s = np.sum(grl)
    j_sq = (np.array(range(grl.shape[1]))+1)**2
    i_sq = (np.array(range(grl.shape[0]))+1)**2

    # props
    SRE = np.sum((g_row/s)/j_sq)
    LRE = np.sum((g_row/s)*j_sq)
    GLN = np.sum(np.sum(grl, axis=1)**2)/np.sum(grl)
    RLN = np.sum(np.sum(grl, axis=0)**2)/np.sum(np.ravel(grl))
    RP = np.sum(np.ravel(grl))/(img.size/pix_dist)
    LGLRE = np.sum((r_col/s)/i_sq)
    HGLRE = np.sum((r_col/s)*i_sq)
    SRLGLE = np.sum([grl[i-1, j-1]/((i**2)*(j**2)) for i in range(1, grl.shape[0]+1) f
    SRHGLE = np.sum([grl[i-1, j-1]*(i**2)/(j**2) for i in range(1, grl.shape[0]+1) for
    LRLGLE = np.sum([grl[i-1, j-1]*(j**2)/(i**2) for i in range(1, grl.shape[0]+1) for
    LRHGLE = np.sum([grl[i-1, j-1]*(j**2)*(i**2) for i in range(1, grl.shape[0]+1) for

    out_l = [SRE, LRE, GLN, RLN, RP,LGLRE, HGLRE, SRLGLE, SRHGLE, LRLGLE, LRHGLE]

    return out_l
```

4 Spectral Texture Features (SP)

Functions generate ring and wedge masks to be applied to input transformed images.

4.1 FPS (Fourier power spectrum)

Angular and radial sums of image wedges are used as features.

Output features are labelled as `SPFPS_R_T_` where the R stands for radius and the T stands for the theta used in wedge generation.

The radius for the wedge generation is dependent on the image size. In general 5 distances are calculated from image center to edge. The angles used are: 0, 45, 90, 135

4.1.1 Refs:

ref: Weszka, J. S., Dyer, C. R., & Rosenfeld, A. (1976). Comparative Study of Texture Measures for Terrain Classification. IEEE Transactions on Systems, Man and Cybernetics, SMC-6(4), 269–285. `http://doi.org/10.1109/TSMC.1976.5408777`

`http://ieeexplore.ieee.org.libproxy1.nus.edu.sg/xpl/articleDetails.jsp?arnumber=5408777`

`http://desidoc.drdo.gov.in/ojs/index.php/dsj/article/view/3951/4437`

4.2 Correlation Feats (CF)

The 2d correlation function from scipy is used to generate auto-correlation images.

Standard statistics of these images are used as features. The output features are named as: `SPCF_x_sum` where sum can be replaced by other properties as below.

Properties	Code
Sum	sum
Mean	mean
Median	med
Maximum	max
Minimum	min
Standard Deviation	std
Kurtosis	kur

4.3 GWT (Gabor Wavelet Transform)

From Ref 1: "GWT features (F33–F74): Gabor filter provides useful texture descriptors by using multi-scale features estimated at different scales and orientations. The 2D- GWT, considering three scales $(0, 1, 2)$ and seven orientations (22.5∘,45∘, 67.5∘,90∘, 112.5∘, 135∘, 157.5∘), result in a group of $(73 = 21)$ wavelets.When this group of Gabor filters family of 21 wavelets is convolved with a given ROI image, a set of 21 feature images are obtained. The real parts of Gabor filter family of twenty one feature images obtained. From these 21 feature images, mean and standard deviation are computed as texture descriptors resulting in (21 feature images×2 statistical parameters = 42) features for each ROI"

Used Theta	22.5, 45, 67.5, 90, 112.5, 135, 157.5
Used Scale	1, 2, 3
Frequency	0.05, 0.25, 0.4

The gabor kernels are generated using above parameters. Each of these is convolved with the image and the mean and variance of the generated image are used as feature.

The features are labelled at `SPGWT_T_x_S_y_mean_` where T stands for theta and S for scale. The theta is multiplied by 10 as dots cause problems for programming. The theta 22.5 thus becomes T_{225}.

Properties	Code
Mean	mean
Variance	var

4.3.1 Refs:

1. Lee, C.-C., & Chen, S.-H. (2006). Gabor Wavelets and SVM Classifier for Liver Diseases Classiflcation from CT Images. In 2006 IEEE International Conference on Systems, Man and Cybernetics (Vol. 1, pp. 548–552). IEEE. http://doi.org/10.1109/ICSMC.2006.384441

2. Tutorial on Gabor Filters http://mplab.ucsd.edu/tutorials/gabor.pdf

3. http://scikit-image.org/docs/dev/auto_examples/plot_gabor.html

4.4 Code:

```
from skimage.draw import circle
import numpy as np
import math
from scipy import ndimage as ndi
from scipy.stats import kurtosis
from skimage.filters import gabor_kernel
from scipy.signal import correlate2d
from helper_func import an_df_filler


def an_get_spec_feats(feat_frame, row_num, crop_im):
    out_spec1, lbl1 = an_fourier_feats(crop_im)
    out_spec1[np.isnan(out_spec1)] = 0
    feat_frame = an_df_filler(feat_frame, row_num, out_spec1, lbl1)

    out_spec2, lbl2 = an_gabor_feats(crop_im)
    out_spec2[np.isnan(out_spec2)] = 0
    feat_frame = an_df_filler(feat_frame, row_num, out_spec2, lbl2)

    out_spec3, lbl3 = an_gen_corr_feat(crop_im)
    out_spec3[np.isnan(out_spec3)] = 0
    feat_frame = an_df_filler(feat_frame, row_num, out_spec3, lbl3)


    return feat_frame



# Helper Functions


def an_gen_ring(radii, psd2D):
    img_cen = np.array(psd2D.shape)/2
    ring_masks = np.zeros((len(radii), psd2D.shape[0], psd2D.shape[1]))
    for count, rad in enumerate(radii):
        idx_img = np.zeros(psd2D.shape)
        rr1, cc1 = circle(img_cen[0], img_cen[1], rad, shape=psd2D.shape)
        rr2, cc2 = circle(img_cen[0], img_cen[1], rad-5, shape=psd2D.shape)
        idx_img[rr1, cc1] = 1
        idx_img[rr2, cc2] = 0

        ring_masks[count, :, :] = idx_img

    return ring_masks
```

```
def an_gen_wedge(in_theta, psd2D):
    img_cen = np.array(psd2D.shape)/2

    idx_img = np.zeros((len(in_theta), psd2D.shape[0], psd2D.shape[1]))

    for count, t in enumerate(in_theta):
        theta1 = t - (45/2)
        theta2 = t + (45/2)+1

        # math.tan(math.radians(theta1))
#         print([theta1,theta2])
        rw = []
        cw = []

        i_range = range(img_cen[0]-2*img_cen[0], img_cen[0])
        j_range = range(img_cen[1]-2*img_cen[1], img_cen[1])

    #     print(i_range)
        for i in np.array(i_range):
            for j in np.array(j_range):
                co_rat = math.degrees(math.atan2(j, i))
                if theta1 <= co_rat < theta2:
                    rw.append(i+img_cen[0])
                    cw.append(j+img_cen[1])

        idx_img[count, rw, cw] = 1

    return idx_img


# Prop Functions

def an_fourier_feats(crop_im):
    fft_im = np.fft.fftshift(np.fft.fft2(crop_im))

    # Calculate a 2D power spectrum
    psd2D = np.abs(fft_im) ** 2
    img_cen = np.array(psd2D.shape) / 2
    # radii = range(5, int(np.sqrt(img_cen[0] ** 2 + img_cen[1] ** 2) + 5), 5)
    radii = np.linspace(5, int(np.sqrt(img_cen[0] ** 2 + img_cen[1] ** 2) + 5), 5)
    in_theta = [0, 45, 90, 135]


    ring_imgs = an_gen_ring(radii, psd2D)
    wedge_imgs = an_gen_wedge(in_theta, psd2D)

    rw_int = [('SPFPS_R' + str(i) + '_T' + str(in_theta[j]) + '_', np.logical_and(im1,
              enumerate(ring_imgs) for j, im2 in enumerate(wedge_imgs)]

    # rw_int = [('SPFPS_R' + str(radii[i]) + '_T' + str(in_theta[j]) + '_', np.logical_
    #           enumerate(ring_imgs) for j, im2 in enumerate(wedge_imgs)]

    #     print([rw_int[i][1] for i,im in enumerate(rw_int)])
    rw_sums = np.ravel([np.sum(psd2D[rw_int[i][1]]) for i, im in enumerate(rw_int)])
    out_col_nm = [rw_int[i][0] for i, im in enumerate(rw_int)]
    # len(rw_sums)

    return rw_sums, out_col_nm


def compute_feats(image, kernels, out_col_nm):
    feats = np.zeros((len(kernels), 2), dtype=np.double)
    out_nm = []
    for k, kernel in enumerate(kernels):
        filtered = ndi.convolve(image, kernel, mode='wrap')
        feats[k, 0] = filtered.mean()
        feats[k, 1] = filtered.var()
        out_nm.append(out_col_nm[k] + 'mean')
        out_nm.append(out_col_nm[k] + 'var')
    return np.ravel(feats), np.ravel(out_nm)


def an_gabor_feats(crop_im):
    # prepare filter bank kernels
    kernels = []
    out_col_nm = []
    # for theta in range(4):
```

```
    for theta in [22.5, 45, 67.5, 90, 112.5, 135, 157.5]:
        #     theta = theta / 4. * np.pi
        theta1 = np.radians(theta)
        for sigma in ([1, 2, 3]):  # sigma is scale
            for frequency in ([0.05, 0.25, 0.4]):
                kernel = np.real(gabor_kernel(frequency, theta=theta1,
                                              sigma_x=sigma, sigma_y=sigma))
                kernels.append(kernel)
                out_col_nm.append('SPGWT_T' + str(int(theta * 10)) + '_S' + str(sigma)

    feats, out_col_nm = compute_feats(crop_im, kernels, out_col_nm)
    #     print(feats.shape)
    #     print(feats)
    gab_feats = np.ravel(feats)

    #     print(gab_feats)

    return gab_feats, out_col_nm


def an_gen_corr_feat(crop_im):
    corr_im = correlate2d(crop_im, crop_im, mode='full', boundary='symm')
    img_cen = np.array(corr_im.shape) / 2
    # radii = range(5, int(np.sqrt(img_cen[0] ** 2 + img_cen[1] ** 2) + 5), 5)
    radii = np.linspace(5, int(np.sqrt(img_cen[0] ** 2 + img_cen[1] ** 2) + 5), 5)
    corr_ring_masks = an_gen_ring(radii, corr_im) > 0

    pro_out_feat = [([np.sum(corr_im[im]), np.mean(corr_im[im]),
                      np.median(corr_im[im]), np.max(corr_im[im]),
                      np.min(corr_im[im]),
                      np.std(corr_im[im]), kurtosis(corr_im[im])],
                     ['SPCF_' + str(i) + '_sum_', 'SPCF_' + str(i) + '_mean_',
                      'SPCF_' + str(i) + '_med_', 'SPCF_' + str(i) + '_max_',
                      'SPCF_' + str(i) + '_min_',
                      'SPCF_' + str(i) + '_std_', 'SPCF_' + str(i) + '_kur_']) for i,

    out_feat = np.reshape([f[0] for f in pro_out_feat], -1)
    out_lbl = np.reshape([f[1] for f in pro_out_feat], -1)
    return out_feat, out_lbl
```

5 Spatial Filtering Based Texture Features (SF)

Law's Texture Features

1. Level detection L5 = [1 4 6 4 1]
2. Edge detection E5 = [-1 -2 0 2 1]
3. Spot detection S5 = [-1 0 2 0 -1]
4. Wave detection W5 = [-1 2 0 -2 -1]
5. Ripple detection R5 = [1 -4 6 -4 1]

Twenty five 2-D Laws' masks for vector length of 5

	L5	E5	S5	W5	R5
L5	L5L5	E5L5	S5L5	W5L5	R5L5
E5	L5E5	E5E5	S5E5	W5E5	R5E5
S5	L5S5	E5S5	S5S5	W5S5	R5S5
W5	L5W5	E5W5	S5W5	W5W5	R5W5
R5	L5R5	E5R5	S5R5	W5R5	R5R5

The approach discussed in the ref 1 has been extended to Law's vectors of length 3,5,7,9 as discussed in ref 2 In brief, the vector combinations are used to generate 2d matrices which are then convolved with the image. Total of 42 Law's masks are used and mean, SD, Kurtosis and Entropy are output as features.

Properties	Code
Standard Deviation	StDev
Skewness	Skew
Kurtosis	Kurt
Entropy	Entropy

The output features are labelled as `SF_L5_R5_StDev` Where the L5_{R5} is a placeholder for combinations and the StDev is placeholder for properties.

The total set of Law's vectors used is below:

Vector	Code	
[1, 2, 1],	L3	
		0, 1
,	E3	
		-2, -1
,	S3	
		4, 6, 4, 1
,	L5	
		-2, 0, 2, 1
,	E5	
		0, 2, 0, -1
,	S5	
		2, 0, -2, -1
,	W5	
		-4, 6, -4, 1
,	R5	
		6, 15, 20, 15, 6, 1
,	L7	
		-4, -5, 0, 5, 4, 1
,	E7	
		-2, 1, 4, 1, -2, -1
,	S7	
		8, 28, 56, 70, 56, 28, 8, 1
,	L9	
		4, 4, -4, -10, -4, 4, 4, 1
,	E9	
		0, -4, 0, 6, 0, -4, 0, 1
,	S9	
		-4, 4, -4, -10, 4, 4, -4, 1
,	W9	
		-8, 28, -56, 70, -56, 28, -8, 1
	R9	

5.0.1 Refs:

1. Rachidi, M., Marchadier, A., Gadois, C., Lespessailles, E., Chappard, C., & Benhamou, C. L. (2008). Laws' masks descriptors applied to bone texture analysis: An innovative and discriminant tool in osteoporosis. Skeletal Radiology, 37(6), 541–548. `http://doi.org/10.1007/s00256-008-0463-2`

2. Ismail, A., & Mahmoud, A. (2016). Image Feature Detectors and Descriptors (Vol. 630). `http://doi.org/10.1007/978-3-319-28854-3`

5.0.2 Code:

```
import numpy as np
import scipy
from an_glcm_props import an_glcm_entropy
from helper_func import an_df_filler


def an_laws_feats(feat_frame, row_num, crop_im):
    pro_law_vecs = [[1, 2, 1],  # L3
                    [-1, 0, 1],  # E3
                    [1, -2, -1],  # S3
                    [1, 4, 6, 4, 1],  # L5
                    [-1, -2, 0, 2, 1],  # E5
                    [-1, 0, 2, 0, -1],  # S5
                    [-1, 2, 0, -2, -1],  # W5
                    [1, -4, 6, -4, 1],  # R5
                    [1, 6, 15, 20, 15, 6, 1],  # L7
                    [-1, -4, -5, 0, 5, 4, 1],  # E7
                    [-1, -2, 1, 4, 1, -2, -1],  # S7
                    [1, 8, 28, 56, 70, 56, 28, 8, 1],  # L9
                    [1, 4, 4, -4, -10, -4, 4, 4, 1],  # E9
                    [1, 0, -4, 0, 6, 0, -4, 0, 1],  # S9
                    [1, -4, 4, -4, -10, 4, 4, -4, 1],  # W9
                    [1, -8, 28, -56, 70, -56, 28, -8, 1]  # R9
                    ]

    pro_pro_lbl = ['L3', 'E3', 'S3', 'L5', 'E5', 'S5', 'W5', 'R5', 'L7', 'E7', 'S7', '

    law_vec_sets = [[0, 2], [3, 7], [8, 10], [11, 15]]  # define groups
    pro_laws_param = []
    out_lbl = []
    for f in law_vec_sets:
        law_vecs = np.array([np.atleast_2d(xi) for xi in pro_law_vecs[f[0]:f[1] + 1]])
        pro_lbl = [pro_pro_lbl[i] for i, j in enumerate(pro_law_vecs[f[0]:f[1] + 1])]
        for i, vec1 in enumerate(law_vecs):
            for j, vec2 in enumerate(law_vecs):
                if i == j:
                    # print(pro_[i,j])
                    law_masks = [vec1 * vec2.T]
                    TR = np.squeeze(an_laws_TEI(crop_im, law_masks))
                    TR = np.ravel(TR)
                    pro_laws_param.append(
                        [np.std(TR), scipy.stats.skew(TR), scipy.stats.kurtosis(TR), a
                    lb = ['StDev', 'Skew', 'Kurt', 'Entropy']
                    out_lbl.append(['SF_' + pro_lbl[i] + '_' + pro_lbl[j] + '_' + l +

                if j > i:
                    # print([i,j])
                    law_masks = [vec1 * vec2.T, vec2 * vec1.T]
                    TEI_imgs = an_laws_TEI(crop_im, law_masks)
                    TR = np.squeeze((TEI_imgs[0] + TEI_imgs[0]) / 2)
                    TR = np.ravel(TR)
                    pro_laws_param.append(
                        [np.std(TR), scipy.stats.skew(TR), scipy.stats.kurtosis(TR), a
                    lb = ['StDev', 'Skew', 'Kurt', 'Entropy']
                    out_lbl.append(['SF_' + pro_lbl[i] + '_' + pro_lbl[j] + '_' + l +

    laws_param = np.reshape(pro_laws_param, -1)
    out_lbl = np.reshape(out_lbl, -1)
    # print(np.array(out_lbl).shape)
    # print(laws_param.shape)
    # laws_param.shape
    feat_frame = an_df_filler(feat_frame, row_num, laws_param, out_lbl)

    return feat_frame


# Helper functions


def an_laws_TI(crop_im, law_masks):
    return np.array([np.array(scipy.signal.convolve2d(crop_im,in_mask, boundary='fill'


def an_laws_TEI(crop_im, law_masks):
    conv_imgs = an_laws_TI(crop_im, law_masks)
    corr_fac = [7, 7]
    img_shape1 = conv_imgs[0].shape[0]-corr_fac[0]
    img_shape2 = conv_imgs[0].shape[1]-corr_fac[1]
    out_TEI = np.zeros((conv_imgs.shape[0], img_shape1-corr_fac[0],img_shape2-corr_fac
    for k, in_img in enumerate(conv_imgs):
        a = np.array([np.sum(np.abs(in_img[i-7:i+8, j-7:j+8]))
                      for i in range(corr_fac[0], img_shape1) for j in range(corr_fac[
        out_TEI[k, :, :] = np.reshape(a, [img_shape1-corr_fac[0],img_shape2-corr_fac[0]

    return out_TEI
```

6 Spatial Diversity Features (SD)

From Refs: "The concept of diversity comes fromthe field of ecology and is one of its central themes. Diversity represents a measure of how species are present in the environment and, more specifically, in different habitats and communities. The analysis of diversity considers concepts like population, diffusion, and diversification and is performed by means of indices that analyze the presence of a species within a given environment. To apply diversity analysis to pattern recognition, it is first necessary to express the data to be analyzed (in this case, images) in terms of population, species, and related concepts."

"population P is defined as the set of all N_p pixels in the image,while the gray level values constitute the different species $s(i)$; the total number of species is S, which can be smaller than 256 if some gray level values are not present in the image. The number of individuals of a species $s(i)$ is the number $n(i)$ of pixels in the image with a given gray level. Finally, the relative frequency for a species is defined as: $p(i) = n(i)/N_p$. Given this model it is possible to exploit the indices commonly used to measure population diversity and apply them to pattern recognition."

Several derivative features are extracted as described in the reference.

Properties (index)	Code
Shannon–Wiener	'$\mathrm{SD}_{\mathrm{Shann\,Wi}\_}$'
McIntosh	'$\mathrm{SD}_{\mathrm{McIntosh}\_}$'
Brillouin	'$\mathrm{SD}_{\mathrm{Brillouin}\_}$'
total diversity	'$\mathrm{SD}_{\mathrm{Tot\,Diver}\_}$'
Simpson	'$\mathrm{SD}_{\mathrm{Simpson}}$'
Berger–Parker	'SD_ BergerParker_'
J	'$\mathrm{SD}_{\mathrm{J}\_}$'
Ed	'$\mathrm{SD}_{\mathrm{Ed}\_}$'
Hill	'$\mathrm{SD}_{\mathrm{Hill}\_}$'
Buzas–Gibson	'$\mathrm{SD}_{\mathrm{BuzasGibson}\_}$'
Camargo	'$\mathrm{SD}_{\mathrm{Camargo}\_}$'

Details of the Properties are as below

6.0.3 Shannon–Wiener index:

This value increases with higher heterogeneity levels.

$$H = -\sum_{i=0}^{S-1} p(i) ln p(i)$$

6.0.4 McIntosh index:

$$Mc = \frac{N_p - \sqrt{\sum_{i=0}^{S-1} n(i)^2}}{N_p - \sqrt{N_p}}$$

6.0.5 Brillouin index:

The Brillouin index, yet another measure of diversity, is best suited when the randomness of the population is not guaranteed:

$$Hb = \frac{1}{N_p}\left(ln(N_p!) - \sum_{i=0}^{S-1} ln(n(i)!) \right)$$

6.0.6 total diversity index:

measures species variation

$$Td = \sum_{i=0}^{S-1} \frac{1}{n(i)} (p(i) * (1 - p(i)))$$

6.0.7 Simpson index:

second order statistics

$$Ds = \frac{\sum_{i=0}^{S-1} n(i)(n(i)-1)}{N_p(N_p-1)}$$

6.0.8 Berger–Parker index:

relative frequency of the most frequent species

$$Bp = max(p_i)$$

6.0.9 J index:

The J index, for example, compares the value of the H index for the current population to the same index provided by a maximally diverse population:

$$H' : J = H/H'$$

$$withH' = lnS$$

6.0.10 Ed index:

The same concept applied to the Simpson index leads to the Ed index: Ed=Ds/Ds

6.0.11 Hill index:

The Hill index is defined as:

$$Hill = \frac{\frac{1}{Ds1}}{e^H 1}$$

6.0.12 Buzas–Gibson index:

The Buzas–Gibson index measures the uniformity of the Shannon index.

$$Bg = \frac{e^H}{S}$$

6.0.13 Camargo index:

Camargo index, which compares the relative frequencies of the different species as follows

$$CamIdx = 1 - \left(\sum_{i=0}^{S-1} \sum_{j=i+1}^{S-1} \frac{p(i) - p(j)}{S} \right)$$

6.0.14 Refs

Nanni, L., Brahnam, S., Ghidoni, S., & Menegatti, E. (2015). Improving the descriptors extracted from the co-occurrence matrix using preprocessing approaches. Expert Systems with Applications, 000, 1–12. `http://doi.org/10.1016/j.eswa.2015.07.055`

6.0.15 Code

```
from skimage import img_as_ubyte
import numpy as np
import math
from helper_func import an_df_filler


def an_spa_div(feat_frame, row_num, crop_im):
    img = img_as_ubyte(crop_im)
    Np = img.size

    s = np.unique(img)
    # print(s)
    S = s.size
    # print(S)

    n_i, bin_edges = np.histogram(img, bins=256)
    # Replace 0s with small values
    n_i = n_i.astype(float) + np.finfo(float).eps
    # print(n_i)

    p_i = n_i / float(Np)
    # Replace 0s with small values
    p_i += + np.finfo(float).eps
    # print(p_i)
    H = -np.sum(p_i * np.log(p_i))  # Shannon Wiener index
    Mc = (Np - np.sqrt(np.sum(n_i ** 2))) / Np - np.sqrt(Np)  # McIntosh index
    Hb = (1.0 / Np) * (math.log(np.math.factorial(Np)) - np.sum([math.log(np.math.facto
                                          for i in range(n_i.shape[0])]))  # Brillou
    Td = np.sum((1.0 / n_i) * (p_i * (1 - p_i)))  # total diversity index
    Ds = (np.sum(n_i * (n_i - 1))) / (Np * (Np - 1))  # The Simpson index
    Bp = np.max(p_i)  # Berger Parker index
    Jidx = H / np.log(S)  # J index
    Ed = Ds / np.log(S)  # Ed index
    HillIdx = (1 / (Ds - 1)) / (np.exp(H) - 1)  # Hill index
    Bg = np.exp(H) / S  # Buzas Gibson index
    CamIdx = 1 - np.sum([(p_i[i] - p_i[i + 1]) / S
                         for i in range(p_i.shape[0] - 1)
                         for j in range(i + 1, p_i.shape[0] - 1)])  # Camargo index

    out_l = [H, Mc, Hb, Td, Ds, Bp,
             Jidx, Ed, HillIdx, Bg, CamIdx]

    out_lbl = ['SD_Shann_Wi_', 'SD_McIntosh_', 'SD_Brillouin_', 'SD_Tot_Diver_', 'SD_Si
               'SD_ BergerParker_', 'SD_J_', 'SD_Ed_', 'SD_Hill_', 'SD_BuzasGibson_',

    feat_frame = an_df_filler(feat_frame, row_num, np.ravel(out_l), out_lbl)
    return feat_frame
```

7 Local Binary Patterns (LBP)

From ref: "To calculate the LBP value for a pixel in the grayscale image, we compare the central pixel value with the neighbouring pixel values. We can start from any neighbouring pixel and then we can transverse either in clockwise or anti-clockwise direction but we must use the same order for all the pixels. Since there are 8 neighbouring pixels – for each pixel, we will perform 8 comparisons. The results of the comparisons are stored in a 8-bit binary array.

If the current pixel value is greater or equal to the neighbouring pixel value, the corresponding bit in the binary array is set to 1 else if the current pixel value is less than the neighbouring pixel value, the corresponding bit in the binary array is set to 0."

The mahotas package is used to get the LBP features.

The features are labelled as: `LBP_r_x_pt_y` where r is the radius and pt is the point on the circle. radii of 1, 3, 5, and 10 are used with 10 points.

7.0.16 Ref:

`http://hanzratech.in/2015/05/30/local-binary-patterns.html`

7.0.17 Code:

```
def an_lbp_props(feat_frame, row_num, bg_zero_im):
    radii = [1, 3, 5, 10]
    no_of_points = 10

    out_lbp = np.array([an_lbp_mahotas(bg_zero_im, radius, no_of_points) for radius in
    out_lbl = ['LBP_r' + str(r) + '_pt_' + str(pt) + '_' for r in range(out_lbp.shape[
               range(out_lbp.shape[1])]
    out_lbp = out_lbp.flatten()

    feat_frame = an_df_filler(feat_frame, row_num, out_lbp, out_lbl)
```

8 Threshold Adjacency Statistics (TAS)

From 1:

"Threshold adjacency statistics are generated by first applying a threshold to the image to create a binary image (Figures 1 and 2), with a threshold chosen as follows. The average intensity, , of those pixels with intensity at least 30 is calculated for the image, the cut off 30 chosen as intensities below this value are in general background, and is considered in more detail below in the Testing subsection (an 8-bit grayscale image has pixel has intensities from 0 to 255). The experimental image is then binary thresholded to the range -30 to +30 (Figure 2a'). The range was selected to maximise the visual difference of threshold images for which the localisation images had distinct localisations but were visually similar, as in Figure 1. The following nine statistics were designed to exploit the dissimilarity seen in the threshold images. For each white pixel, the number of adjacent white pixels is counted (Figure 2 (0)-(8)). The first threshold statistic is then the number of white pixels with no white neighbours; the second is the number with one white neighbour, and so forth up to the maximum of eight. The nine statistics are normalised by dividing each by the total number of white pixels in the threshold image. Two other sets of threshold adjacency statistics are also calculated as above, but for binary threshold images with pixels in the ranges -30 to 255 and to 255, giving in total 27 statistics"

From 2:

"We also adapted the threshold adjacency statistic features (tas) from Hamilton et al. (16) to a parameter-free version. The original features depended on a manually controlled-two-step binarization of the image. For the first step, we use the Ridler–Calvard algorithm to identify a threshold instead of a fixed threshold (17). The second binarization step involves finding those pixels that fall into a given interval such as $[M, +M]$, where is the average pixel value of the above-threshold pixel and M is a margin (set to 30 in the original paper). We set M to the standard deviation of the above threshold pixels. We call these parameter-free tas."

The mahotas package is used to get TAS features. specifically the parameter free TAS function is used.

8.0.18 Refs:

1. Original TAS: Hamilton, N. a, Pantelic, R. S., Hanson, K., & Teasdale, R. D. (2007). Fast automated cell phenotype image classification. BMC Bioinformatics, 8, 110. `http://doi.org/10.1186/1471-2105-8-110`

2. parameter free: Ahmed, A., Arnold, A., Coelho, L. P., Kangas, J., Sheikh, A. S., Xing, E., . . . Murphy, R. F. (2010). Structured literature image finder: Parsing text and figures in biomedical literature. Journal of Web Semantics, 8(2-3), 151–154. `http://doi.org/10.1016/j.websem.2010.04.002`

8.0.19 Code:

```
def an_tas_props(feat_frame, row_num, crop_im):
    an_tas = mahotas.features.pftas(skimage.img_as_uint(crop_im))

    feat_frame = an_df_filler(feat_frame, row_num, an_tas, ['TAS'])

    return feat_frame
```

9 Zernike Moments

Zernike moments are used to compress images and are robust compared to alternatives. An image with zero background is used so that only the nuclear pixels are used for encoding. The mahotas package is used to calculate zernike moments at radii: 1, 2, 3, 5, 10.

9.0.20 Refs:

http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/SHUTLER3/node11.html

9.0.21 Code:

```
def an_zernike_props(feat_frame, row_num, bg_zero_im):
    radii = [1, 2, 3, 5, 10]
    out_zernike = np.array([an_zernike_mahotas(bg_zero_im, radius) for radius in radii
    # print(out_zernike.shape)
    out_lbl = ['ZER_r'+str(r)+'_pt_'+str(pt)+'_' for r in range(out_zernike.shape[0]) 
    # print(out_lbl)
    out_zernike = out_zernike.flatten()

    feat_frame = an_df_filler(feat_frame, row_num, out_zernike, out_lbl)
    return feat_frame
```

10 Save images as hdf5

10.1 Description

Function and driver code to save images as hdf5 files. Folder containing `.tiff` files are, the image class and the file-names are given as inputs. Multiple folders can also be combined into a single hdf5 file.

10.2 Code

```
def an_mk_hdf5(root_root, in_name_list, db_name, write_path):
    file_names = np.array([glob(os.path.join(root+'*.tif')) for root in root_root])
    tot_files = np.sum([len(fn) for fn in file_names])

    i_siz = imread(file_names[0][0]).shape
    i_dtype = imread(file_names[0][0]).dtype

    with h5py.File(write_path+db_name) as f:
        imgs = f.require_dataset('images', shape=(tot_files, i_siz[0], i_siz[1]), dtyp
        dt = h5py.special_dtype(vlen=bytes)
        celltype = f.create_dataset('CellType', (tot_files, 1), dtype=dt) # dtype="S10

        count = 0
        for j, ctype in enumerate(in_name_list):
            for i, fn in enumerate(file_names[j]):
                im = imread(fn)
                imgs[count] = im
                celltype[count] = ctype
                count += 1
    print('HDF5 written to: ' + write_path + db_name)
    return

from an_img_io.an_hdf5_io import an_mk_hdf5

write_path = '/media/data/Aneesh/Source Images/img_pickles/all_combined_hdf5/hdf5_sets/
# db_name = 'mcf7_Load_deconv_20160623.hdf5'


# root = '','','/media/data/Aneesh/Source Images/img_pickles/Fibroblasts/TNF/'
root_root = [
# MCF7
    ['MCF7_1_20160320_20160309_combi', ['MCF7'], ['/media/data/Aneesh/Source Images/img
                                                  # '/media/data/Aneesh/Source Images/i
                                                  ]],
]


for r in root_root:
    # [r[0] for r in root_root]
    an_mk_hdf5(r[2], r[1], r[0]+'.hdf5', write_path)
```

11 Bibliography

bibliography:refs.bib