ДОДАТОК А

Вихідний текст реалізованих алгоритмів аналізу зображень

```
import pandas as pd

df = pd.read_csv(filepath_or_buffer='https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data',
                 header=None, sep=',')

df.columns = ['sepal_len', 'sepal_wid', 'petal_len', 'petal_wid', 'class']
df.dropna(how="all", inplace=True)  # drops the empty line at file-end

df.tail()

# split data table into data X and class labels y

X = df.ix[:, 0:4].values
y = df.ix[:, 4].values

# plotting histograms with online library pyplot

traces = []

legend = {0: False, 1: False, 2: False, 3: True}

colors = {'Iris-setosa': 'rgb(31, 119, 180)',
          'Iris-versicolor': 'rgb(255, 127, 14)',
          'Iris-virginica': 'rgb(44, 160, 44)'}

# for col in range(4):
#     for key in colors:
#         traces.append(Histogram(x=X[y==key, col],
#                                 opacity=0.75,
#                                 xaxis='x%s' %(col+1),
#                                 marker=Marker(color=colors[key]),
#                                 name=key,
#                                 showlegend=legend[col]))
#
# data = Data(traces)
#
# layout = Layout(barmode='overlay',
#                 xaxis=XAxis(domain=[0, 0.25], title='sepal length (cm)'),
#                 xaxis2=XAxis(domain=[0.3, 0.5], title='sepal width (cm)'),
#                 xaxis3=XAxis(domain=[0.55, 0.75], title='petal length (cm)'),
#                 xaxis4=XAxis(domain=[0.8, 1], title='petal width (cm)'),
#                 yaxis=YAxis(title='count'),
#                 title='Distribution of the different Iris flower features')
```

```
#
# fig = Figure(data=data, layout=layout)
# py.iplot(fig)

# Standardizing
from sklearn.preprocessing import StandardScaler

X_std = StandardScaler().fit_transform(X)

# PCA in scikit-learn
from sklearn.decomposition import PCA as sklearnPCA
sklearn_pca = sklearnPCA(n_components=2)
Y_sklearn = sklearn_pca.fit_transform(X_std)


print(Y_sklearn)

METHOD = 'uniform'
plt.rcParams['font.size'] = 9


def plot_circle(ax, center, radius, color):
    circle = plt.Circle(center, radius, facecolor=color, edgecolor='0.5')
    ax.add_patch(circle)


def plot_lbp_model(ax, binary_values):
    """Draw the schematic for a local binary pattern."""
    # Geometry spec
    theta = np.deg2rad(45)
    R = 1
    r = 0.15
    w = 1.5
    gray = '0.5'

    # Draw the central pixel.
    plot_circle(ax, (0, 0), radius=r, color=gray)
    # Draw the surrounding pixels.
    for i, facecolor in enumerate(binary_values):
        x = R * np.cos(i * theta)
        y = R * np.sin(i * theta)
        plot_circle(ax, (x, y), radius=r, color=str(facecolor))

    # Draw the pixel grid.
    for x in np.linspace(-w, w, 4):
        ax.axvline(x, color=gray)
        ax.axhline(x, color=gray)
```

```
    # Tweak the layout.
    ax.axis('image')
    ax.axis('off')
    size = w + 0.2
    ax.set_xlim(-size, size)
    ax.set_ylim(-size, size)


fig, axes = plt.subplots(ncols=5, figsize=(7, 2))

titles = ['flat', 'flat', 'edge', 'corner', 'non-uniform']

binary_patterns = [np.zeros(8),
                   np.ones(8),
                   np.hstack([np.ones(4), np.zeros(4)]),
                   np.hstack([np.zeros(3), np.ones(5)]),
                   [1, 0, 0, 1, 1, 1, 0, 0]]

for ax, values, name in zip(axes, binary_patterns, titles):
    plot_lbp_model(ax, values)
    ax.set_title(name)

import numpy as np
from scipy import ndimage as ndi
import matplotlib.pyplot as plt

from skimage.filters import sobel
from skimage.segmentation import slic, join_segmentations
from skimage.morphology import watershed
from skimage.color import label2rgb, rgb2gray, rgb2hed
from skimage import data, img_as_float
from skimage import io

coins = img_as_float(data.coins())
# MGM
imageRGB                                                                    =
io.imread("L:\mgm\DISSER\project_python\TS_12_05_17_31_07_x2.png")
#image = rgb2gray(imageRGB) * 100
image_hed = rgb2hed(imageRGB)
#coins = image.astype(np.uint8)     #convrt to uint8
coins = image_hed[:, :, 0]


# make segmentation using edge-detection and watershed
edges = sobel(coins)
markers = np.zeros_like(coins)
```

```
foreground, background = 1, 2
#markers[coins < 30.0 / 255] = background
#markers[coins > 150.0 / 255] = foreground
markers[coins < 20.0 ] = background
markers[coins > 60.0 ] = foreground

ws = watershed(edges, markers)
seg1 = ndi.label(ws == foreground)[0]

# make segmentation using SLIC superpixels
seg2 = slic(coins, n_segments=117, max_iter=160, sigma=1, compactness=0.75,
          multichannel=False)

# combine the two
segj = join_segmentations(seg1, seg2)

# show the segmentations
fig, axes = plt.subplots(ncols=4, figsize=(9, 2.5), sharex=True, sharey=True,
subplot_kw={'adjustable':'box-forced'})
axes[0].imshow(coins, cmap=plt.cm.gray, interpolation='nearest')
axes[0].set_title('Image')

color1 = label2rgb(seg1, image=coins, bg_label=0)
axes[1].imshow(color1, interpolation='nearest')
axes[1].set_title('Sobel+Watershed')

color2 = label2rgb(seg2, image=coins, image_alpha=0.5)
axes[2].imshow(color2, interpolation='nearest')
axes[2].set_title('SLIC superpixels')

color3 = label2rgb(segj, image=coins, image_alpha=0.5)
axes[3].imshow(color3, interpolation='nearest')
axes[3].set_title('Join')

for ax in axes:
   ax.axis('off')
fig.subplots_adjust(hspace=0.01, wspace=0.01, top=1, bottom=0, left=0, right=1)
#plt.show()

fig2, axes2 = plt.subplots(ncols=2, figsize=(9, 2.5), sharex=True, sharey=True,
subplot_kw={'adjustable':'box-forced'})
axes2[0].imshow(edges, cmap=plt.cm.gray, interpolation='nearest')
axes2[0].set_title('Edges')

axes2[1].imshow(markers, cmap=plt.cm.spectral, interpolation='nearest')
axes2[1].set_title('Markers')
fig2.subplots_adjust(hspace=0.01, wspace=0.01, top=1, bottom=0, left=0, right=1)
```

```
plt.show()

#io.imsave("result.png", result)


from __future__ import print_function

import matplotlib.pyplot as plt
import numpy as np
from scipy import ndimage as ndi

from skimage import data
from skimage.util import img_as_float
from skimage.filters import gabor_kernel


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


def match(feats, ref_feats):
    min_error = np.inf
    min_i = None
    for i in range(ref_feats.shape[0]):
        error = np.sum((feats - ref_feats[i, :])**2)
        if error < min_error:
            min_error = error
            min_i = i
    return min_i


# prepare filter bank kernels
kernels = []
for theta in range(4):
    theta = theta / 4. * np.pi
    for sigma in (1, 3):
        for frequency in (0.05, 0.25):
            kernel = np.real(gabor_kernel(frequency, theta=theta,
                                          sigma_x=sigma, sigma_y=sigma))
            kernels.append(kernel)
```

```
shrink = (slice(0, None, 3), slice(0, None, 3))
brick = img_as_float(data.load('brick.png'))[shrink]
grass = img_as_float(data.load('grass.png'))[shrink]
wall = img_as_float(data.load('rough-wall.png'))[shrink]
image_names = ('brick', 'grass', 'wall')
images = (brick, grass, wall)

# prepare reference features
ref_feats = np.zeros((3, len(kernels), 2), dtype=np.double)
ref_feats[0, :, :] = compute_feats(brick, kernels)
ref_feats[1, :, :] = compute_feats(grass, kernels)
ref_feats[2, :, :] = compute_feats(wall, kernels)

print('Rotated images matched against references using Gabor filter banks:')

print('original: brick, rotated: 30deg, match result: ', end='')
feats = compute_feats(ndi.rotate(brick, angle=190, reshape=False), kernels)
print(image_names[match(feats, ref_feats)])

print('original: brick, rotated: 70deg, match result: ', end='')
feats = compute_feats(ndi.rotate(brick, angle=70, reshape=False), kernels)
print(image_names[match(feats, ref_feats)])

print('original: grass, rotated: 145deg, match result: ', end='')
feats = compute_feats(ndi.rotate(grass, angle=145, reshape=False), kernels)
print(image_names[match(feats, ref_feats)])


def power(image, kernel):
    # Normalize images for better comparison.
    image = (image - image.mean()) / image.std()
    return np.sqrt(ndi.convolve(image, np.real(kernel), mode='wrap')**2 +
                   ndi.convolve(image, np.imag(kernel), mode='wrap')**2)

# Plot a selection of the filter bank kernels and their responses.
results = []
kernel_params = []
for theta in (0, 1):
    theta = theta / 4. * np.pi
    for frequency in (0.1, 0.4):
        kernel = gabor_kernel(frequency, theta=theta)
        params = 'theta=%d,\nfrequency=%.2f' % (theta * 180 / np.pi, frequency)
        kernel_params.append(params)
        # Save kernel and the power image for each image
        results.append((kernel, [power(img, kernel) for img in images]))

fig, axes = plt.subplots(nrows=5, ncols=4, figsize=(5, 6))
```

```
plt.gray()

fig.suptitle('Image responses for Gabor filter kernels', fontsize=12)

axes[0][0].axis('off')

# Plot original images
for label, img, ax in zip(image_names, images, axes[0][1:]):
    ax.imshow(img)
    ax.set_title(label, fontsize=9)
    ax.axis('off')

for label, (kernel, powers), ax_row in zip(kernel_params, results, axes[1:]):
    # Plot Gabor kernel
    ax = ax_row[0]
    ax.imshow(np.real(kernel), interpolation='nearest')
    ax.set_ylabel(label, fontsize=7)
    ax.set_xticks([])
    ax.set_yticks([])

    # Plot Gabor responses with the contrast normalized for each filter
    vmin = np.min(powers)
    vmax = np.max(powers)
    for patch, ax in zip(powers, ax_row[1:]):
        ax.imshow(patch, vmin=vmin, vmax=vmax)
        ax.axis('off')

plt.show()


imageRGB                                                                    =
io.imread("L:\mgm\DISSER\project_python\TS_12_05_17_31_07_patch_x2reduced.
jpg")
#image = color.rgb2gray(imageRGB) * 100
#sarraster = image.astype(np.uint8)
sarraster = color.rgb2gray(imageRGB)

# Create rasters to receive texture and define filenames
contrastraster = np.copy(sarraster)
contrastraster[:] = 0

dissimilarityraster = np.copy(sarraster)
dissimilarityraster[:] = 0

homogeneityraster = np.copy(sarraster)
homogeneityraster[:] = 0
```

```
energyraster = np.copy(sarraster)
energyraster[:] = 0

correlationraster = np.copy(sarraster)
correlationraster[:] = 0

ASMraster = np.copy(sarraster)
ASMraster[:] = 0

# Create figure to receive results
fig = plt.figure()
fig.suptitle('GLCM Textures')

# In first subplot add original SAR image
ax = plt.subplot(241)
plt.axis('off')
ax.set_title('Original Image')
plt.imshow(sarraster, cmap='gray')

for i in range(sarraster.shape[0]):
    print i,
    for j in range(sarraster.shape[1]):

        # windows needs to fit completely in image
        if i > (contrastraster.shape[0] - 4) or j > (contrastraster.shape[0] - 4):
            continue

        # Define size of moving window
        glcm_window = sarraster[i - 3: i + 4, j - 3: j + 4]
        # Calculate GLCM and textures
        glcm = greycomatrix(glcm_window, [1], [0], symmetric=True, normed=True)

        # Calculate texture and write into raster where moving window is centered
        contrastraster[i, j] = greycoprops(glcm, 'contrast')
        dissimilarityraster[i, j] = greycoprops(glcm, 'dissimilarity')
        homogeneityraster[i, j] = greycoprops(glcm, 'homogeneity')
        energyraster[i, j] = greycoprops(glcm, 'energy')
        correlationraster[i, j] = greycoprops(glcm, 'correlation')
        ASMraster[i, j] = greycoprops(glcm, 'ASM')
        glcm = None
        glcm_window = None

texturelist = {1: 'contrast', 2: 'dissimilarity', 3: ' homogeneity', 4: 'energy', 5:
'correlation', 6: 'ASM'}
for key in texturelist:
    ax = plt.subplot(2, 3, key)
    plt.axis('off')
```

```
    ax.set_title(texturelist[key])
    plt.imshow(eval(texturelist[key] + "raster"), cmap='gray')

plt.show()
```

ДОДАТОК Б ДОВІДКА ПРО ВИКОРИСТАННЯ

ДОДАТОК В

Копія публікації