# 07_svhn_preprocessing

September 29, 2021

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
import os
import sys
import tarfile
import tensorflow as tf
from IPython.display import display, Image
from scipy import ndimage
import h5py
from PIL import Image
import PIL.Image as Image
%matplotlib inline
```

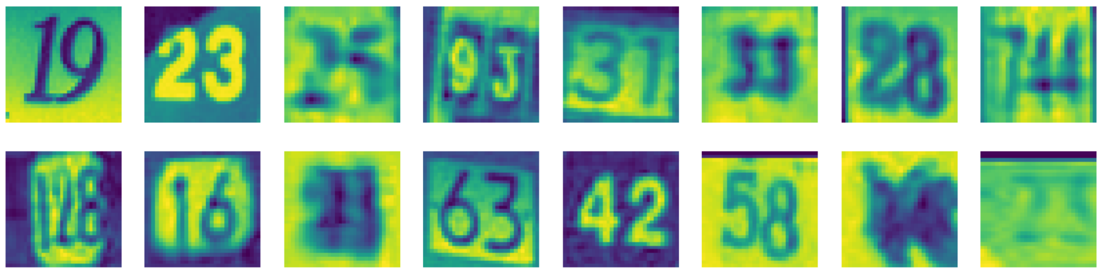
[2]:
```
class DigitStructFile:
    def __init__(self, inf):
        self.inf = h5py.File(inf, 'r')
        self.digitStructName = self.inf['digitStruct']['name']
        self.digitStructBbox = self.inf['digitStruct']['bbox']

    def getName(self, n):
        return ''.join([chr(c[0]) for c in self.inf[self.digitStructName[n][0]].
 ↪value])

    def bboxHelper(self, attr):
        if (len(attr) > 1):
            attr = [self.inf[attr.value[j].item()].value[0][0]
                    for j in range(len(attr))]
        else:
            attr = [attr.value[0][0]]
        return attr

    def getBbox(self, n):
        bbox = {}
        bb = self.digitStructBbox[n].item()
        bbox['height'] = self.bboxHelper(self.inf[bb]["height"])
        bbox['label'] = self.bboxHelper(self.inf[bb]["label"])
        bbox['left'] = self.bboxHelper(self.inf[bb]["left"])
        bbox['top'] = self.bboxHelper(self.inf[bb]["top"])
        bbox['width'] = self.bboxHelper(self.inf[bb]["width"])
        return bbox

    def getDigitStructure(self, n):
        s = self.getBbox(n)
        s['name'] = self.getName(n)
        return s

    def getAllDigitStructure(self):
        return [self.getDigitStructure(i) for i in range(len(self.
↪digitStructName))]

    def getAllDigitStructure_ByDigit(self):
        pictDat = self.getAllDigitStructure()
        result = []
        structCnt = 1
        for i in range(len(pictDat)):
            item = {'filename': pictDat[i]["name"]}
            figures = []
            for j in range(len(pictDat[i]['height'])):
                figure = {}
                figure['height'] = pictDat[i]['height'][j]
                figure['label'] = pictDat[i]['label'][j]
                figure['left'] = pictDat[i]['left'][j]
                figure['top'] = pictDat[i]['top'][j]
                figure['width'] = pictDat[i]['width'][j]
                figures.append(figure)
            structCnt = structCnt + 1
            item['boxes'] = figures
            result.append(item)
        return result
```

[3]:
```
def generate_dataset(data, folder):

    dataset = np.ndarray([len(data), 32, 32, 1], dtype='float32')
    labels = np.ones([len(data), 6], dtype=int) * 10
    for i in np.arange(len(data)):
        filename = data[i]['filename']
        fullname = os.path.join(folder, filename)
        im = Image.open(fullname)
        boxes = data[i]['boxes']
        num_digit = len(boxes)
        labels[i, 0] = num_digit
        top = np.ndarray([num_digit], dtype='float32')
        left = np.ndarray([num_digit], dtype='float32')
        height = np.ndarray([num_digit], dtype='float32')
        width = np.ndarray([num_digit], dtype='float32')
        for j in np.arange(num_digit):
            if j < 5:
                labels[i, j+1] = boxes[j]['label']
                if boxes[j]['label'] == 10:
                    labels[i, j+1] = 0
            else:
                print('#', i, 'image has more than 5 digits.')
            top[j] = boxes[j]['top']
            left[j] = boxes[j]['left']
            height[j] = boxes[j]['height']
            width[j] = boxes[j]['width']

        im_top = np.amin(top)
        im_left = np.amin(left)
        im_height = np.amax(top) + height[np.argmax(top)] - im_top
        im_width = np.amax(left) + width[np.argmax(left)] - im_left

        im_top = np.floor(im_top - 0.1 * im_height)
        im_left = np.floor(im_left - 0.1 * im_width)
        im_bottom = np.amin([np.ceil(im_top + 1.2 * im_height), im.size[1]])
        im_right = np.amin([np.ceil(im_left + 1.2 * im_width), im.size[0]])

        im = im.crop((im_left, im_top, im_right, im_bottom)
                     ).resize([32, 32], Image.ANTIALIAS)
        im = np.dot(np.array(im, dtype='float32'),
                    [[0.2989], [0.5870], [0.1140]])
        mean = np.mean(im, dtype='float32')
        std = np.std(im, dtype='float32', ddof=1)
        if std < 1e-4:
            std = 1.
        im = (im - mean) / std
        dataset[i, :, :, :] = im[:, :, :]
    return dataset, labels
```

```
[ ]: for folder in ['train', 'test', 'extra']:
         print(folder)
         path = Path('images', 'svhn', folder)
         target = path  / 'digitStruct.mat'
         dsf = DigitStructFile(target)
         data = dsf.getAllDigitStructure_ByDigit()
         dataset, labels = generate_dataset(data, path)
         dataset = dataset.reshape(dataset.shape[0], -1)
         with pd.HDFStore('images/svnh/data.h5') as store:
             store.put(f'{folder}/data', pd.DataFrame(dataset))
             store.put(f'{folder}/labels', pd.DataFrame(labels))
```

[4]:
```
with pd.HDFStore('images/svhn/data.h5') as store:
    X_train = store['train/data'].values.reshape(-1, 32, 32, 1)
```

[7]:
```
fig, axes = plt.subplots(nrows=2, ncols=8, figsize=(20, 5))
axes = axes.flatten()
for i, ax in enumerate(axes):
    ax.imshow(np.squeeze(X_train[i]))
    ax.axis('off')
fig.savefig('images/svhn', dpi=300);
```



[ ]:
```
```