MachineVision HW3 - Image Thinning

Author

- 00657113 Chao, Jun-Kai (Compeador)

- Done at 2020/06/02

External Link

- Video on Youtube (https://youtu.be/WdNDWXqJH54)
- Source on Github (https://github.com/ken1882/NTOU2020_MachineVision/tree/master/hw3_thinning)

Abstract

This program uses Zhang and Suen thinning algorithm to implement the image skeleton aka thinning. With the uses of the convolutional computing it boosts the performance compared to pure loop-based python code.

Primary Code Explaination

Pre-processing

First the image will convert to grayscale and padding proper value to do the adaptive threshold.

```
inputs   = cv2.cvtColor(inputs,cv2.COLOR_BGR2GRAY)
PADDING_WIDTH = 30
paddings = tf.constant([[PADDING_WIDTH, PADDING_WIDTH], [PADDING_WIDTH, PADDING_W
pad_inputs = tf.pad(copy(inputs), paddings, "SYMMETRIC")
binary = AdaptiveThreshold(61,-8)(inputs, pad_inputs)
```

then the binarization will be applied for later thinning

```
x  = tf.nn.conv2d(pad_input, self.filters, strides=1, padding="VALID")
th = x - self.C
return np.where(ori_input > th, 1, 0)
```

Rules implement

In order to fit the rule into convolutional calculation, two following kernel is designed to do the job.

```
filters1 = np.array([
  [1, 1, 1],
  [1, 0, 1],
  [1, 1, 1]
]).reshape(3, 3, 1, 1)
filters2 = np.array([
  [  1,  2,  4],
  [128,  0,  8],
  [ 64, 32, 16]
]).reshape(3, 3, 1, 1)
```

First kernel is to fit the first rule in both pass of ZS algorithm, the number foreground neighbor of the central pixel.
Second kernel using the bitmask to store the position information of the neighbor pixels.

So the position rule can be done with following code with binary operations to determine the `S(p)`

```
def _match_transition(mat_ele):
  val  = mat_ele
  fst  = val & 1
  val += fst * 256 # N7 -> N0
  lst  = cur = fst
  val  = val >> 1
  cnt  = 0
  while val > 0:
    cur = val & 1
    if lst == 0 and cur == 1:
      cnt += 1
    lst = cur
    val = val >> 1
  if cur == 0 and fst == 1:
    cnt += 1
  return cnt == 1

match_transition = np.vectorize(_match_transition)
```

```
band = tf.bitwise.bitwise_and

# pass 1
x2 = tf.nn.conv2d(x, self.filters1, strides=1, padding="SAME")
x3 = tf.nn.conv2d(x, self.filters2, strides=1, padding="SAME")
x2 = np.where(((x2 >= 2) & (x2 <= 6)), 1, 0)
x3 = np.where(
  (
    (
      (
        ((band(x3,2) == 0) | (band(x3,8) == 0) | (band(x3,32) == 0)) &
        ((band(x3,128) == 0) | (band(x3,8) == 0) | (band(x3,32) == 0))
      )
    ) & match_transition(x3)
  ), 1, 0
)
x = np.where(((x == 1) & (x2 == 1) & (x3 == 1)), 0, x)
# pass 2
x2 = tf.nn.conv2d(x, self.filters1, strides=1, padding="SAME")
x3 = tf.nn.conv2d(x, self.filters2, strides=1, padding="SAME")
x2 = np.where(((x2 >= 2) & (x2 <= 6)), 1, 0)
x3 = np.where(
  (
    (
      (
        ((band(x3,2) == 0) | (band(x3,8) == 0) | (band(x3,128) == 0)) &
        ((band(x3,2) == 0) | (band(x3,32) == 0) | (band(x3,128) == 0))
      )
    ) & match_transition(x3)
  ), 1, 0
)
x = np.where(((x == 1) & (x2 == 1) & (x3 == 1)), 0, x)
```

Finally the convolution of `x3` will actually calculate the data and matching whether the pixel has matched the rest rules of the ZS algorithm; with both pass 1 ans 2, until the image no longer has pixel be ripped.

References

- https://www.tensorflow.org/api_docs/python/tf/nn/conv2d (https://www.tensorflow.org/api_docs/python/tf/nn/conv2d)
- https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html (https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html)
- http://ccy.dd.ncu.edu.tw/~chen/course/vision/ch7/第七單元 細線化與骨架抽取.pdf (http://ccy.dd.ncu.edu.tw/~chen/course/vision/ch7/%E7%AC%AC%E4%B8%83%E5%96%AE%E5%85%83%20%E7%B4%B0%E7%B7%9A%E5%8C%96%E8%88%87%E9%AA%A8%E6%9E%B6%E6%8A%BD%E5%8F%96.pdf)
- https://www.tensorflow.org/api_docs/python/tf/pad (https://www.tensorflow.org/api_docs/python/tf/pad)
- https://rosettacode.org/wiki/Zhang-Suen_thinning_algorithm (https://rosettacode.org/wiki/Zhang-Suen_thinning_algorithm)