Hi, Boris and Chris. Sorry for the delay.

I have worked on constructing a mathematical model of the cognitive algorithm for several days. Work is not finished yet, but I get certain thoughts and propositions, and I would like to discuss them.

I believe that the mathematical logic that you are realized in the code could be optimized in the future in a more optimal way. That is why, first of all, I would like to understand what do we have as input and output data. Browsing the results of the data processing in the debug mode seems not very convenient to me, so I have added saving processed results in the "csv" file. As far as I understand those results for 1D algorithm are stored into frame_of_patterns_ list.

```
with open('frame_of_patterns_.txt', 'w') as f:
    for item in frame_of_patterns_:
        f.write("%s\n" % item)
```

I got a file with a size about 2.6 M. Content looks like this:

frame_of_patterns_.txt ×

```
[CP(L=2, I=196, D=22, M=-22.5), CP(L=34, I=3058, D=-50, M=607), CP(L=1, I=76, D=17, M=-4),
M=26), CP(L=4, I=574, D=-72, M=-16), CP(L=22, I=2201, D=-29, M=352), CP(L=5, I=295, D=10, M
I=984, D=48, M=143), CP(L=1, I=122, D=-20, M=-2), CP(L=4, I=375, D=-44, M=32), CP(L=3, I=35
I=393, D=-18, M=55), CP(L=1, I=126, D=26, M=-7), CP(L=14, I=1858, D=14, M=263), CP(L=1, I=1
CP(L=4, I=557, D=-10, M=38), CP(L=1, I=149, D=18, M=-4), CP(L=9, I=1578, D=46, M=136), CP(L
M=-13), CP(L=4, I=339, D=20, M=58), CP(L=3, I=416, D=79, M=-83), CP(L=6, I=1135, D=11, M=12
D=40, M=-16), CP(L=1, I=165, D=15, M=2), CP(L=2, I=333, D=-10, M=-22), CP(L=2, I=266, D=-21
D=-17, M=-187), CP(L=2, I=359, D=20, M=18), CP(L=1, I=162, D=-19, M=-2), CP(L=7, I=1065, D=
I=700, D=19, M=-58), CP(L=2, I=206, D=-21, M=34), CP(L=3, I=407, D=63, M=-52), CP(L=8, I=13
CP(L=3, I=354, D=78, M=-79), CP(L=2, I=358, D=27, M=8), CP(L=3, I=389, D=-70, M=-41), CP(L=
CP(L=7, I=710, D=-66, M=-169), CP(L=16, I=1928, D=16, M=286), CP(L=3, I=263, D=-69, M=-44),
M=449), CP(L=4, I=475, D=-60, M=-32), CP(L=9, I=894, D=34, M=116), CP(L=3, I=241, D=-72, M=
M=171), CP(L=8, I=586, D=-20, M=-146), CP(L=14, I=1130, D=25, M=202), CP(L=1, I=79, D=-16,
D=-3, M=8), CP(L=1, I=72, D=19, M=-2), CP(L=19, I=1870, D=14, M=254), CP(L=2, I=173, D=14,
D=-14, M=62), CP(L=1, I=103, D=22, M=-2), CP(L=4, I=413, D=-6, M=59), CP(L=9, I=1095, D=56,
```

For more convenient data analysis I have started from the more compact representation of results.

I added saving some initial variables like _p, _d, and _m into "csv" file:

	A	B	C
1	**# p=**	**d=**	**m=**
2	103	12	3
3	116	13	2
4	123	7	8
5	126	3	12
6	129	3	12
7	129	0	15
8	128	-1	14
9	124	-4	11
10	128	4	11
11	130	2	13
12	132	2	13
13	134	2	13
14	136	2	13
15	132	-4	11
16	127	-5	10
17	122	-5	10
18	121	-1	14
19	120	-1	14
20	121	1	14
21	122	1	14
22	123	1	14
23	122	-1	14
24	121	-1	14
25	113	-8	7
26	106	-7	8
27	107	1	14
28	117	10	5
29	120	3	12
30	113	-7	8
31	106	-7	8

Next, I would like to propose how to process them in a more simple and fast way than is realized now. This is part of the existed code with my addons for saving intermediate processing results:

```
# Read image directly numpy Matrix format
# 0 Read in gray picture, 1 read in color picture
img = cv2.imread("raccoon.jpg", 0).astype("int16")

ave = 15  # |difference| between pixels that coincides with average value of Pm
ave_min = 2  # for m defined as min |d|: smaller?
# min M for initial incremental-range comparison(t_), higher cost than der_comp?
ave_M = 50
ave_D = 5  # min |D| for initial incremental-derivation comparison(d_)
ave_nP = 5  # average number of sub_Ps in P, to estimate intra-costs? ave_rdn_inc = 1 + 1 / ave_nP # 1.2
ave_rdm = 0.5  # average dm / m, to project bi_m = m * 1.5
init_y = 0  # starting row, the whole frame doesn't need to be processed

with open("sample.csv", "w") as csvFile:
    pixel_ = img[-1, :]  # read first line of pixels
    np.savetxt("pixel_2.csv", pixel_, delimiter=";", fmt='%d')
    dert_ = []
    __p, _p = pixel_[0:2]  # each prefix '_' denotes prior
    _d = _p - __p  # initial comparison
    _m = ave - abs(_d)
    # project _m to bilateral m, first dert is for comp_P only?
    dert_.append(Cdert(p=__p, d=None, m=(_m + _m / 2)))
    fieldnames = ("# p=", "d=", "m=")
    write = csv.writer(csvFile, delimiter=";")
    write.writerow(fieldnames)

    for p in pixel_[2:]:  # pixel p is compared to prior pixel _p in a row
        d = p - _p
        # initial match is inverse deviation of |difference|
        m = ave - abs(d)
        # write.writerow([_d, _m])  # +++
        write.writerow([_p, _d, _m])  # +++
        # pack dert: prior p, prior d, bilateral match
        dert_.append(Cdert(p=_p, d=_d, m=m + _m))
        # write.writerow([d, m, _p, _d, (m + _m)])  # +++
        _p, _d, _m = p, d, m
    # unilateral d, forward-project last m to bilateral m
    dert_.append(Cdert(p=_p, d=_d, m=(_m + _m / 2)))
```

Next, I would like to propose how to process them in a more simple and fast way than is realized now. This is part of the existed code with my addons for saving intermediate processing results:

Instead of processing every pair of pixels in the loop using native python tools, all the array that represents an image row could be processed simultaneously using the NumPy module, as far as NumPy is much faster in such operations. We just make a cyclic shift and calculate the vector of differences for all pixels simultaneously. After that next string calculates all the matches in the same way:

```
pixel_shifted_ = np.roll(pixel_, 1)
d = (pixel_ - pixel_shifted_)
m = ave - abs(d[1:-1])
np.savetxt("sample2.csv", np.array([pixel_[1:-1], d[1:-1], m]).T, delimiter=";", fmt='%d', header='p=; d=; m=')
```

I have checked, the results of data processing are the same, but in a more simple and faster way. It only begins, I guess that in the 2D mode same approach could give even more.