time-analysis

August 12, 2022

1 Interactive image manipulation using morphological trees and spline-based skeletons

1.1 Time execution analysis

imports

[1]:
```
import numpy as np
from imageio.v2 import imwrite, imread

from os import listdir
from os.path import isfile, join

from pandas import DataFrame, read_csv, concat

import matplotlib.pyplot as plt

from IPython.display import HTML


plt.figure(figsize=(20, 20))
```

[1]:
```
<Figure size 1440x1440 with 0 Axes>
```

```
<Figure size 1440x1440 with 0 Axes>
```

1.2 Table

Recover images

[2]:
```
img_filenames = [f for f in listdir("./images") if f[-3:] == "pgm"]
npixels = []
dims = []
for img_filename in img_filenames:
    f = imread(join("./images/", img_filename))
    npixels.append(f.shape[0] * f.shape[1])
    dims.append(f"{f.shape[1]} x {f.shape[0]}")

idx = np.argsort(npixels).astype(int)
```

```
img_filenames = np.array(img_filenames)
npixels = np.array(npixels)
dims = np.array(dims)
```

[3]:
```
df = DataFrame(
    {"filename": img_filenames[idx],
     "number of pixels": npixels[idx],
     "dimension": dims[idx]})
df
```

[3]:

	filename	number of pixels	dimension
0	house.pgm	65536	256 x 256
1	bridge.pgm	65536	256 x 256
2	camera.pgm	65536	256 x 256
3	bird.pgm	65536	256 x 256
4	3.pgm	102400	320 x 320
5	w6.pgm	143000	500 x 286
6	2.pgm	244400	400 x 611
7	output6.pgm	248400	540 x 460
8	Fig.17.pgm	254400	600 x 424
9	mandrill.pgm	262144	512 x 512
10	barb.pgm	262144	512 x 512
11	boat.pgm	262144	512 x 512
12	zelda.pgm	262144	512 x 512
13	goldhill.pgm	262144	512 x 512
14	lena.pgm	262144	512 x 512
15	washsat.pgm	262144	512 x 512
16	peppers.pgm	262144	512 x 512
17	w2.pgm	270000	600 x 450
18	Fig.5.pgm	280000	560 x 500
19	mountain.pgm	307200	640 x 480

1.2.1 Load runtime tables.

The cell below contains a code to create a clickable cell of the runtime table. However, We have not found a way to

[4]:
```
# code inspired by
# https://datascientyst.com/create-clickable-link-pandas-dataframe-jupyterlab/

filenames = df["filename"].values
index = {}
for i in range(df.shape[0]):
    index[filenames[i]] = i

def make_clickable(img):
```

```
    i = index[img]
    return f'<a target="_blank" href="http://localhost:8888/lab/tree/
↪plot_img_rutime.ipynb?row={i}">plot link</a>'
    #return f'[plot link](./plot_img_rutime.ipynb)'
```

In the table below, we have the rutime of the morphological tree display and skeleton computation step performed by the proposed tool. The time was computed after the execution of the DMD pipeline functionality (clicking the button "run"). The parameters of the DMD pipeline are set up as the default values except by the layer parameter (L, or the maximum number of grey level kept). In this experiment, we used different values of L: 10, 45, 80, 115, 150, 185, 220, and 255 and registered the runtime (in milliseconds) and the number of morphological tree nodes for each image. Thus, the columns L_time and L_nnodes ($L \in \{10, 45, 80, 115, 150, 185, 220, 255\}$) represent (i) the runtime of skeleton computation and morphological tree display; and (ii) number of morphological tree nodes, respectively. Both measurments are registered after performing the DMD pipeline with the default parameter and number of layers "L".

[5]:
```
time = read_csv("./data/benchmark.csv", sep=";").drop(["filename"], axis=1)
nnodes = read_csv("./data/number_of_nodes.csv", sep=";").drop(["filename"],␣
↪axis=1)

times_nnodes = time.join(nnodes, lsuffix="_time", rsuffix="_nnodes")

df = df.join(times_nnodes)
#df["plot link"] = df.apply(lambda row: make_clickable(row[0]), axis=1)
df
```

[5]:

	filename	number of pixels	dimension	10_time	45_time	80_time	\
0	house.pgm	65536	256 x 256	3474	11177	16493	
1	bridge.pgm	65536	256 x 256	3629	12444	18643	
2	camera.pgm	65536	256 x 256	3013	10577	18331	
3	bird.pgm	65536	256 x 256	1707	7404	13019	
4	3.pgm	102400	320 x 320	6092	20370	32032	
5	w6.pgm	143000	500 x 286	6380	20346	32040	
6	2.pgm	244400	400 x 611	341	341	345	
7	output6.pgm	248400	540 x 460	23607	83987	132092	
8	Fig.17.pgm	254400	600 x 424	3851	15052	15162	
9	mandrill.pgm	262144	512 x 512	32848	106203	151696	
10	barb.pgm	262144	512 x 512	18393	61354	94832	
11	boat.pgm	262144	512 x 512	27464	98566	148104	
12	zelda.pgm	262144	512 x 512	13779	51205	81776	
13	goldhill.pgm	262144	512 x 512	22591	82438	125045	
14	lena.pgm	262144	512 x 512	14236	50175	78195	
15	washsat.pgm	262144	512 x 512	24266	24269	24453	
16	peppers.pgm	262144	512 x 512	16219	62298	103922	
17	w2.pgm	270000	600 x 450	14253	50884	77307	
18	Fig.5.pgm	280000	560 x 500	2437	2446	2455	

19	mountain.pgm	307200	640 x 480	37219	120227	170334

	115_time	150_time	185_time	220_time	255_time	10_nnodes	45_nnodes	\
0	26025	25882	24516	25431	24694	221	595	
1	23719	28185	31403	31628	31530	286	982	
2	25499	30053	30227	29822	29905	174	576	
3	18437	20910	20843	20879	21375	76	299	
4	48327	48966	48400	48392	48825	309	1018	
5	41390	55858	55987	56163	55924	238	748	
6	340	345	340	348	339	5	5	
7	165483	185790	187987	185666	185914	485	1701	
8	15083	15098	15102	15152	15040	64	246	
9	186754	210300	209933	208238	212074	716	2257	
10	120913	143082	151915	151603	151885	392	1289	
11	176039	199279	199910	199702	204392	579	2075	
12	114488	127165	127757	127178	127467	282	1047	
13	161177	194266	193742	194152	193965	465	1694	
14	100985	135812	139341	136024	135907	287	1013	
15	24255	24429	24206	24357	24424	533	533	
16	257907	301901	204117	204239	203136	325	1233	
17	100314	120234	136347	135366	137138	268	940	
18	2536	2443	2427	2451	2432	33	33	
19	170034	169865	170325	169828	171900	643	2013	

	80_nnodes	115_nnodes	150_nnodes	185_nnodes	220_nnodes	255_nnodes
0	742	937	937	937	937	937
1	1427	1780	2022	2249	2249	2249
2	861	1111	1254	1254	1254	1254
3	450	593	632	632	632	632
4	1592	2282	2282	2282	2282	2282
5	1178	1501	1995	1995	1995	1995
6	5	5	5	5	5	5
7	2625	3279	3640	3640	3640	3640
8	246	246	246	246	246	246
9	3163	3872	4269	4269	4269	4269
10	1950	2482	2877	3065	3065	3065
11	3098	3632	4071	4071	4071	4071
12	1653	2279	2530	2530	2530	2530
13	2543	3233	3843	3843	3843	3843
14	1591	2009	2691	2691	2691	2691
15	533	533	533	533	533	533
16	2047	2729	3964	3964	3964	3964
17	1426	1825	2184	2448	2448	2448
18	33	33	33	33	33	33
19	2861	2861	2861	2861	2861	2861

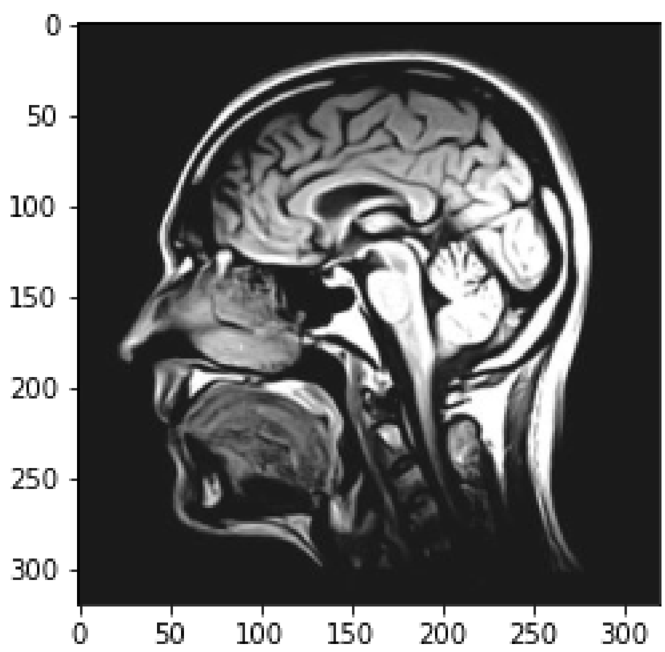
1.3 Plotting

1.3.1 Maximum number of grey levels x runtime

[6]:
```
times_header = ["10_time", "45_time", "80_time", "115_time", "150_time",
                "185_time", "220_time", "255_time"]

nnodes_header = ["10_nnodes", "45_nnodes", "80_nnodes", "115_nnodes",␣
 ↪"150_nnodes",
                 "185_nnodes", "220_nnodes", "255_nnodes"]
```

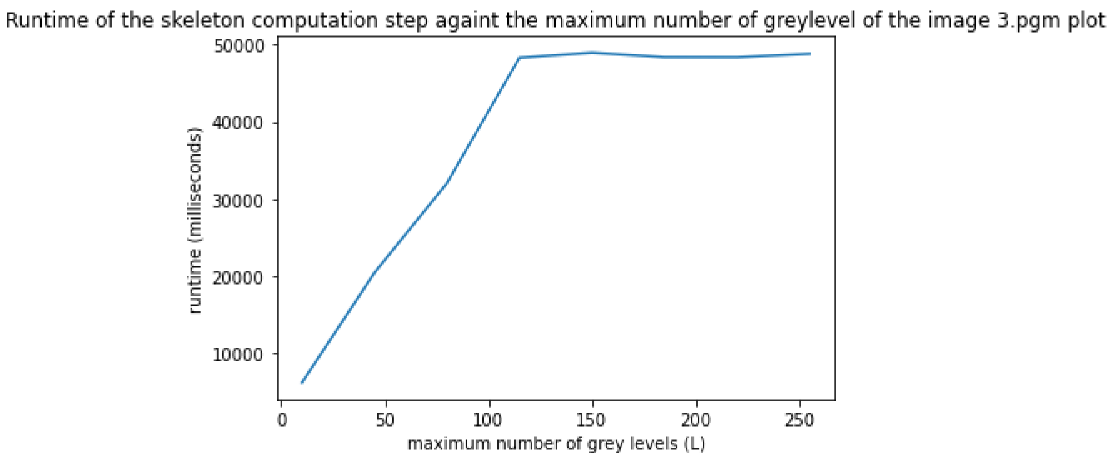
To change the image (row) being plotted, please change the variabel "row" to the index of the image from the table above and run the cells below.

[7]:
```
# L from the DMD pipeline executation (maximum number of grey levels kept)
L = [10, 45, 80, 115, 150, 185, 220, 255]

# index of the image/row in the table
row = 4

# collect the runtime of the image
times = df[times_header].iloc[row]

img = imread(join("./images", df["filename"][row]))

plt.imshow(img, cmap="gray")
plt.show()
```



[8]:
```
#plot
plt.title("Runtime of the skeleton computation step againt the maximum number␣
 ↪of greylevel of the image "
          f"{df['filename'].iloc[row]} plot")
plt.xlabel("maximum number of grey levels (L)")
plt.ylabel("runtime (milliseconds)")
plt.plot(L, times)
```

[8]:
```
[<matplotlib.lines.Line2D at 0x7f2e21ad8ee0>]
```



1.3.2 Image size x runtime

Selecting layer (L from DMD pipeline)

[9]:
```
selected_L_index = 7
selected_L = L[selected_L_index]
selected_L_header = f"{selected_L}_time"
selected_L_header
```

[9]:
```
'255_time'
```

Creating dataset of number of pixels and runtime given a selected L

[10]:
```
times = df[selected_L_header]
npixels_col = df["number of pixels"]
dimension_col = df["dimension"]

npixels_times_df = DataFrame({
    "number of pixels": npixels_col,
    "dimension": dimension_col,
```

```
    "times": times})
npixels_times_df
```

[10]:

	number of pixels	dimension	times
0	65536	256 x 256	24694
1	65536	256 x 256	31530
2	65536	256 x 256	29905
3	65536	256 x 256	21375
4	102400	320 x 320	48825
5	143000	500 x 286	55924
6	244400	400 x 611	339
7	248400	540 x 460	185914
8	254400	600 x 424	15040
9	262144	512 x 512	212074
10	262144	512 x 512	151885
11	262144	512 x 512	204392
12	262144	512 x 512	127467
13	262144	512 x 512	193965
14	262144	512 x 512	135907
15	262144	512 x 512	24424
16	262144	512 x 512	203136
17	270000	600 x 450	137138
18	280000	560 x 500	2432
19	307200	640 x 480	171900

Group the images with the same dimension and using the average runtime

[11]:
```
avg_npixels_times = npixels_times_df.groupby(by=["number of pixels",
    "dimension"], as_index=False).mean()
avg_npixels_times
```

[11]:

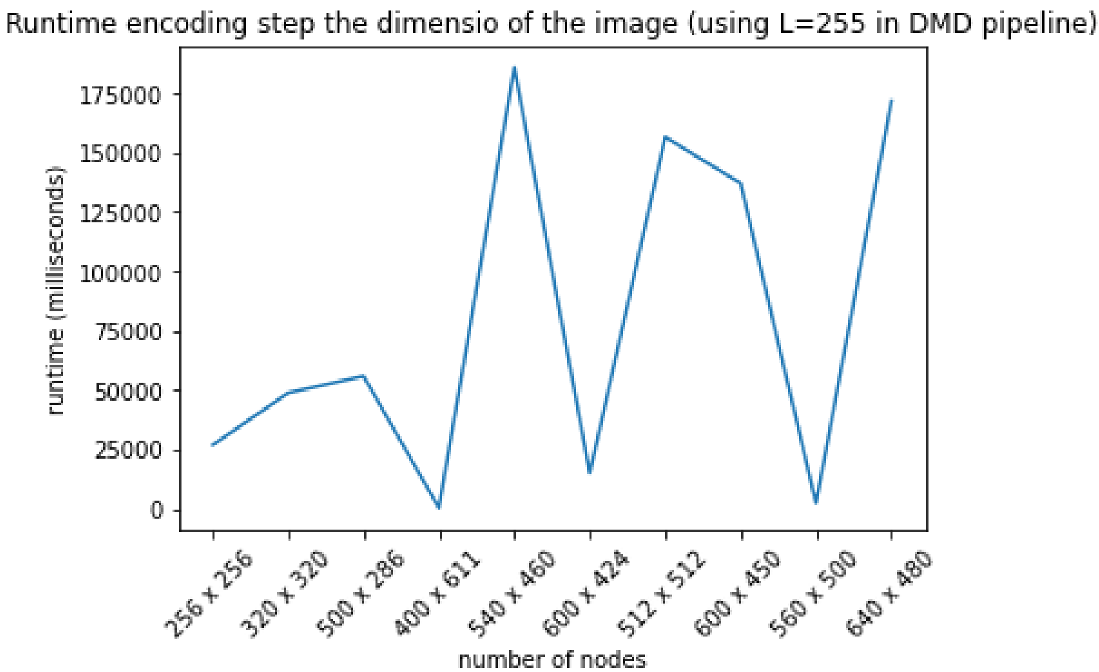
	number of pixels	dimension	times
0	65536	256 x 256	26876.00
1	102400	320 x 320	48825.00
2	143000	500 x 286	55924.00
3	244400	400 x 611	339.00
4	248400	540 x 460	185914.00
5	254400	600 x 424	15040.00
6	262144	512 x 512	156656.25
7	270000	600 x 450	137138.00
8	280000	560 x 500	2432.00
9	307200	640 x 480	171900.00

Plotting

[12]:
```
plt.title(f"Runtime encoding step the dimensio of the image (using
    L={selected_L} in DMD pipeline)")
```

```
plt.xlabel("number of nodes")
plt.ylabel("runtime (milliseconds)")
plt.xticks(rotation = 45)
plt.plot(avg_npixels_times["dimension"], avg_npixels_times["times"])
```

[12]: [<matplotlib.lines.Line2D at 0x7f2e20238f70>]



Removing outliners (number of nodes are always the same or number of nodes for L=10 is the same as the number of nodes for L=255).

[13]:
```
npixels_times_df_no_outliers = npixels_times_df[df["10_nnodes"] !=␣
↪df["255_nnodes"]]
npixels_times_df_no_outliers
```

[13]:

	number of pixels	dimension	times
0	65536	256 x 256	24694
1	65536	256 x 256	31530
2	65536	256 x 256	29905
3	65536	256 x 256	21375
4	102400	320 x 320	48825
5	143000	500 x 286	55924
7	248400	540 x 460	185914
8	254400	600 x 424	15040
9	262144	512 x 512	212074
10	262144	512 x 512	151885

```
11            262144  512 x 512  204392
12            262144  512 x 512  127467
13            262144  512 x 512  193965
14            262144  512 x 512  135907
16            262144  512 x 512  203136
17            270000  600 x 450  137138
19            307200  640 x 480  171900
```

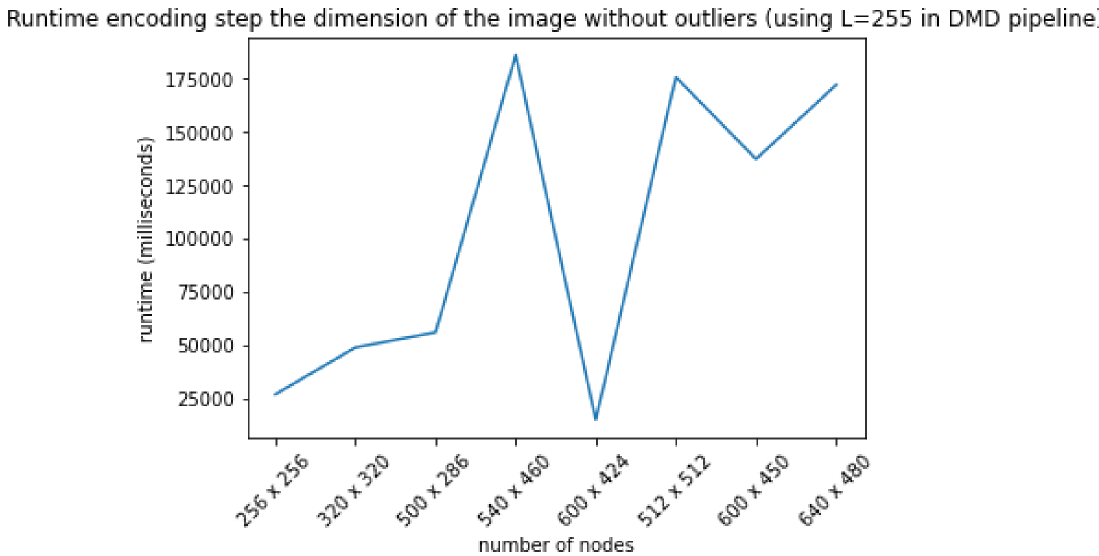
[14]:
```
avg_npixels_times_no_outliers = npixels_times_df_no_outliers.
 ↪groupby(by=["number of pixels", "dimension"], as_index=False).mean()
avg_npixels_times_no_outliers
```

[14]:

	number of pixels	dimension	times
0	65536	256 x 256	26876.000000
1	102400	320 x 320	48825.000000
2	143000	500 x 286	55924.000000
3	248400	540 x 460	185914.000000
4	254400	600 x 424	15040.000000
5	262144	512 x 512	175546.571429
6	270000	600 x 450	137138.000000
7	307200	640 x 480	171900.000000

[15]:
```
plt.title(f"Runtime encoding step the dimension of the image without outliers␣
 ↪(using L={selected_L} in DMD pipeline)")
plt.xlabel("number of nodes")
plt.ylabel("runtime (milliseconds)")
plt.xticks(rotation = 45)
plt.plot(avg_npixels_times_no_outliers["dimension"],␣
 ↪avg_npixels_times_no_outliers["times"])
```

[15]:
```
[<matplotlib.lines.Line2D at 0x7f2e201b4190>]
```



1.3.3 Number of nodes x runtime

[16]:
```
# Get a list of number of nodes
lnnodes = df[nnodes_header].values.ravel()

# Get a list of number of nodes
ltimes = df[times_header].values.ravel()

# Note that  ltimes[index] is the runtime computed by a image whose maxtree has␣
↪lnodes[index] nodes.

# compute average time for experiments with the same number of nodes
times_nodes = DataFrame({"nnodes": lnnodes, "times": ltimes})
avg_times_nodes = times_nodes.groupby(by=["nnodes"], as_index=False).mean()
avg_times_nodes
```

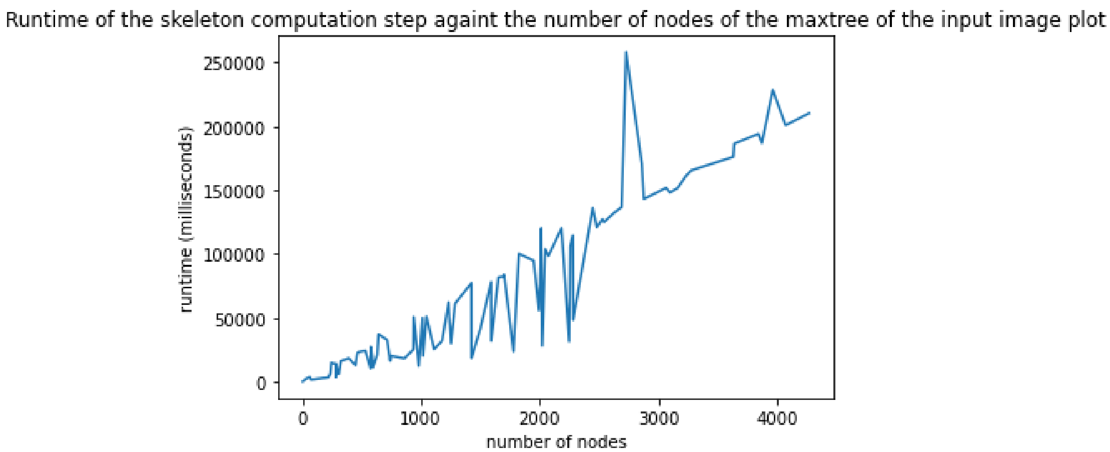
[16]:

	nnodes	times
0	5	342.375
1	33	2453.375
2	64	3851.000
3	76	1707.000
4	174	3013.000
..	…	…
79	3843	194031.250
80	3872	186754.000
81	3964	228348.250
82	4071	200820.750
83	4269	210136.250

[84 rows x 2 columns]

[17]:
```
#plot
plt.title("Runtime of the skeleton computation step againt the number of nodes␣
↪of the maxtree of the input"
          f" image plot")
plt.xlabel("number of nodes")
plt.ylabel("runtime (milliseconds)")
plt.plot(avg_times_nodes["nnodes"], avg_times_nodes["times"])
```

[17]: [<matplotlib.lines.Line2D at 0x7f2e20114f40>]



[ ]:

[ ]: