ImageBench v2 - Benchmarking Tool for Computer Vision Tool Kit

Date	**Updates**	**Doc. Version**	**Authored By**
10-Nov-2020	User Reference for ImageBench	1.0	Mohamed Yusuf
18-Nov-2020	- Added a section for Setup - Algorithm specific configurations - Efficiency & productivity enhancements - Updated *Sampling Scores* matrix - Added *Performance Scores* matrix	1.1	
26-Nov-2020	- Updates in Installation Steps	1.2	
06-Jan-2021	- Updates in Installation Steps	1.3	

1. Description

This is a tool built for computing and benchmarking the scores and performance of various computer vision algorithms for automating the comparisons of large-scale image data sets. The goal is to attain the required degree of the visibility into the image comparison operations driven by CV and provide users with first-hand experience to write automated tests effectively.

2. Purpose

(i) Arrive at similarity scores for large sets of offline images in an easy and seamless manner without any dependencies on automated test runs

(ii) Use the scores to determine the efficacy and veracity of various algorithms against images

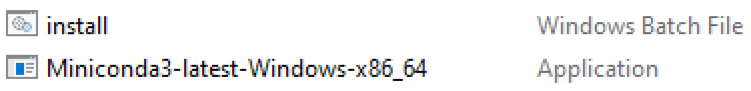
(iii) Find out the performance of each algorithm against various image data set

(iv) Determine the order of the algorithms to be used in actual test runs in cases where it needs

3. Setup

It's pretty simple and straightforward. Two packages need to be installed: a) Miniconda, b) ImageBench

a) Miniconda: Unzip miniconda.zip. Run install.bat. This will install Miniconda in C:\py\miniconda



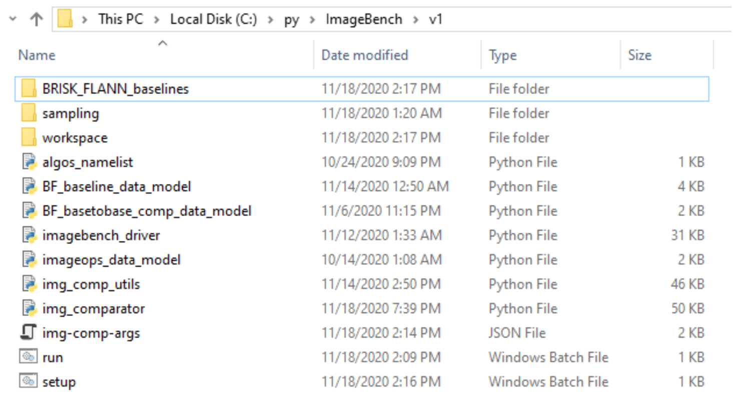
install	Windows Batch File
Miniconda3-latest-Windows-x86_64	Application

<u>Installation Steps:</u>

1. Copy zip files to c:\py
2. Unzip miniconda.zip to C:\py\installer
3. Navigate to c:\py\installer\miniconda. Run Install.bat from the command prompt
4. Wait for two to three minutes. After waiting, you should be able to see c:\py\miniconda with all the necessary files
5. Restart the system
6. Open command prompt. Enter "conda activate". You should be able to see the prompt changing "(base) c:\py". This step confirms that the miniconda installation is successful.
7. Enter "conda deactivate".

b) ImageBench

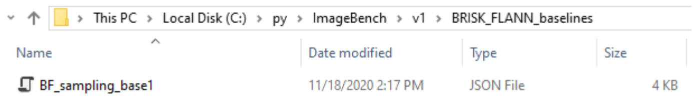
1. Unzip ImageBench.zip to c:\py. The extracted files will be unzipped to c:\py\imagebench
2. Open cmd prompt and navigate to c:\py\imagebench
3. Enter "setup" and wait for a few minutes. This will create the necessary environments.
4. Once setup is done, enter "run". You should be able to see ImageBench sample images are being exercised by the algos with default configurations.


This PC › Local Disk (C:) › py › ImageBench › v1

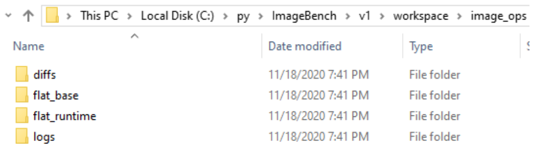
Name	Date modified	Type	Size
BRISK_FLANN_baselines	11/18/2020 2:17 PM	File folder	
sampling	11/18/2020 1:20 AM	File folder	
workspace	11/18/2020 2:17 PM	File folder	
algos_namelist	10/24/2020 9:09 PM	Python File	1 KB
BF_baseline_data_model	11/14/2020 12:50 AM	Python File	4 KB
BF_basetobase_comp_data_model	11/6/2020 11:15 PM	Python File	2 KB
imagebench_driver	11/12/2020 1:33 AM	Python File	31 KB
imageops_data_model	10/14/2020 1:08 AM	Python File	2 KB
img_comp_utils	11/14/2020 2:50 PM	Python File	46 KB
img_comparator	11/18/2020 7:39 PM	Python File	50 KB
img-comp-args	11/18/2020 2:14 PM	JSON File	2 KB
run	11/18/2020 2:09 PM	Windows Batch File	1 KB
setup	11/18/2020 2:16 PM	Windows Batch File	1 KB

BRISK_FLANN_baselines and *workspace* folders are automatically created as a part of setup. By default, *BRISK_FLANN_baselines* folder contains baseline records for the sample images generated from the BRISK-FLANN algo operation. The *workspace* folder contains all artifacts generated from the run operations of algos. They include logs, results, baseline, runtime and diff images.

Given below some reference snapshots that you will see after running the algos:


This PC › Local Disk (C:) › py › ImageBench › v1 › BRISK_FLANN_baselines

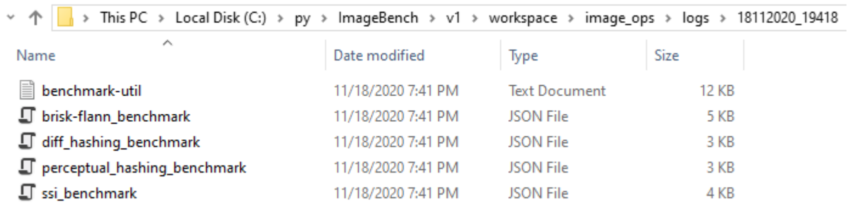
Name	Date modified	Type	Size
BF_sampling_base1	11/18/2020 2:17 PM	JSON File	4 KB


This PC › Local Disk (C:) › py › ImageBench › v1 › workspace › image_ops

Name	Date modified	Type
diffs	11/18/2020 7:41 PM	File folder
flat_base	11/18/2020 7:41 PM	File folder
flat_runtime	11/18/2020 7:41 PM	File folder
logs	11/18/2020 7:41 PM	File folder

This PC › Local Disk (C:) › py › ImageBench › v1 › workspace › image_ops › diffs › 18112020_19418 › BRISK_FLANN

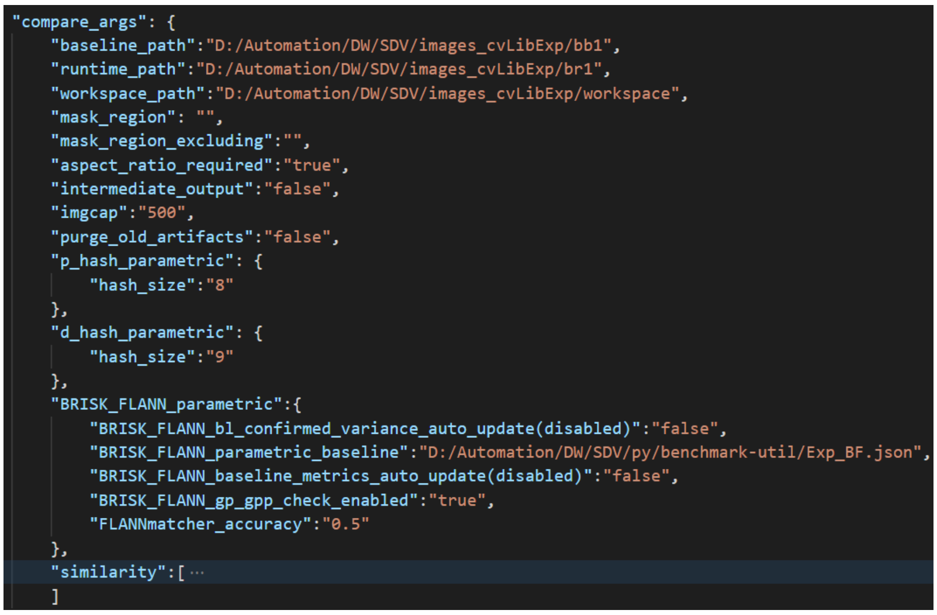
This PC › Local Disk (C:) › py › ImageBench › v1 › workspace › image_ops › diffs › 18112020_19418 › ssi


This PC › Local Disk (C:) › py › ImageBench › v1 › workspace › image_ops › logs › 18112020_19418

Name	Date modified	Type	Size
benchmark-util	11/18/2020 7:41 PM	Text Document	12 KB
brisk-flann_benchmark	11/18/2020 7:41 PM	JSON File	5 KB
diff_hashing_benchmark	11/18/2020 7:41 PM	JSON File	3 KB
perceptual_hashing_benchmark	11/18/2020 7:41 PM	JSON File	3 KB
ssi_benchmark	11/18/2020 7:41 PM	JSON File	4 KB

4. Features and Configurations

For now, all configurations are passed through: ***img-comp-args.json***



```
"compare_args": {
    "baseline_path":"D:/Automation/DW/SDV/images_cvLibExp/bb1",
    "runtime_path":"D:/Automation/DW/SDV/images_cvLibExp/br1",
    "workspace_path":"D:/Automation/DW/SDV/images_cvLibExp/workspace",
    "mask_region": "",
    "mask_region_excluding":"",
    "aspect_ratio_required":"true",
    "intermediate_output":"false",
    "imgcap":"500",
    "purge_old_artifacts":"false",
    "p_hash_parametric": {
        "hash_size":"8"
    },
    "d_hash_parametric": {
        "hash_size":"9"
    },
    "BRISK_FLANN_parametric":{
        "BRISK_FLANN_bl_confirmed_variance_auto_update(disabled)":"false",
        "BRISK_FLANN_parametric_baseline":"D:/Automation/DW/SDV/py/benchmark-util/Exp_BF.json",
        "BRISK_FLANN_baseline_metrics_auto_update(disabled)":"false",
        "BRISK_FLANN_gp_gpp_check_enabled":"true",
        "FLANNmatcher_accuracy":"0.5"
    },
    "similarity":[ ···
    ]
```

A. Feature Index-1

baseline_path, runtime_path:

Specify the baseline and runtime image paths at the root level, regardless of the level of subdirectories. For example: *D:\Automation\DW\SDV\src\test-inputs\images*.

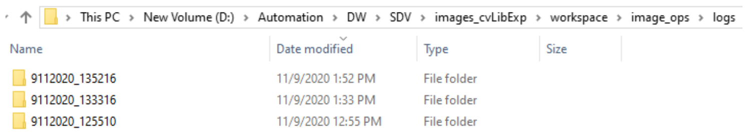
B. Feature Index-2

workspace_path:

workspace_path is where the baseline and runtime images are copied to, but it will be a flattened copy – no subdirectories will be created. Additional subfolders will be auto-created for the organization of the artifacts based on operation sessions.

For example, the *workspace_path* at runtime will create: *logs*, *diffs*(for storing diff files between two images), *flat_base* and *flat_runtime* under *image_ops* subfolder.

Also, *diffs* and *logs* folders will have session specific subfolders to keep log files and diff image files for the corresponding session.



This PC › New Volume (D:) › Automation › DW › SDV › images_cvLibExp › workspace › image_ops › logs

Name	Date modified	Type	Size
9112020_135216	11/9/2020 1:52 PM	File folder	
9112020_133316	11/9/2020 1:33 PM	File folder	
9112020_125510	11/9/2020 12:55 PM	File folder	

During the copy, images that have conflicting names/invalid names are automatically renamed by suffixing the image paths.

C. Feature Index-3

mask_region, mask_region_excluding:

Specify the coordinates of a region in the image, as a tuple, in scenarios where you would want to ignore certain portions of an image for comparison with another image. Ex: 30,25,60,60 stand for x1,y1,x2,y2.

mask_region_excluding, on the other hand, works opposite to *mask_region* i.e. it applies mask on the entire region in an image other than the specified coordinates.

This FI will be useful in SDP/Live time based real-time scenarios, where you would want to ignore the time-scale index, as it would never match with the runtime images.

D. Feature Index-4

aspect_ratio_required:

Need to be “true” when source and runtime image dimensions are different, but need to be resized without sacrificing the image contents in accordance with the respective image’s width and height. However, if this field is “false”, image resizing operations will still be performed under the hood, but that’s raw form of resize i.e. image may be distorted in some cases.

E. Feature Index-5

intermediate_output:

Set to “true” when you want to see additional console output. In some algo operations, this may show transformed/ processed / pre-processed images in separate windows for each image.

F. Feature Index-6

imgcap:

Allows you to specify the maximum number of images to be copied and compared, irrespective of the number of images present in the source and runtime folders. Less testing conducted on the mechanism of the image selection. However, it ensures that the same images are copied from the source and runtime paths to the *flat_base* and *flat_runtime* folders.

G. Feature Index-7

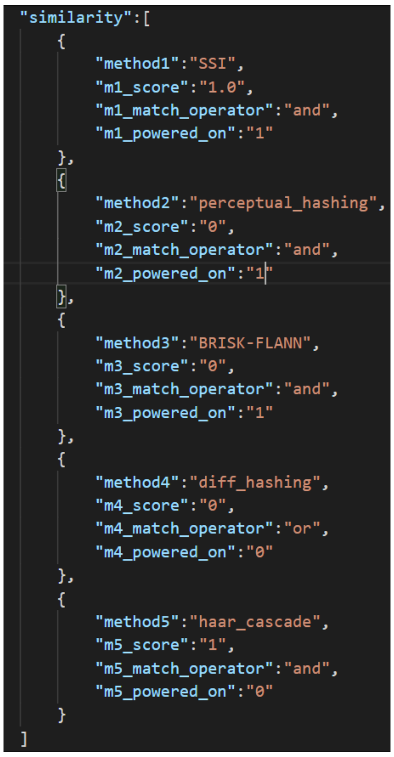
purge_old_artifacts:

Setting this to “true” will remove all the previous sessions’ artifacts generated from the algo operations.

H. Feature Index-8

Algorithms: Order and baseline scores.

You are free to change the order of the algorigthms. Whereas, each algorithm can either be turned on or turned off. Below value of 1 against “*m1_powered_on*” indicates *SSI* is turned on. “*m4_powered_on*” value is set to be 0 indicates that *diff_hashing* is turned off.



```
"similarity":[
    {
        "method1":"SSI",
        "m1_score":"1.0",
        "m1_match_operator":"and",
        "m1_powered_on":"1"
    },
    {
        "method2":"perceptual_hashing",
        "m2_score":"0",
        "m2_match_operator":"and",
        "m2_powered_on":"1"
    },
    {
        "method3":"BRISK-FLANN",
        "m3_score":"0",
        "m3_match_operator":"and",
        "m3_powered_on":"1"
    },
    {
        "method4":"diff_hashing",
        "m4_score":"0",
        "m4_match_operator":"or",
        "m4_powered_on":"0"
    },
    {
        "method5":"haar_cascade",
        "m5_score":"1",
        "m5_match_operator":"and",
        "m5_powered_on":"0"
    }
]
```

Out of the five algorithms listed above, four of them have been implemented in v1.0. Implemented : SSI, perceptual hashing, diff hashing and BRISK-FLANN. The last one – haar cascade will be added in a subsequent version. Until then, keep it at the end of the list.

You can configure the order of the algos as per your need for each run. The scores of each algo listed above are meant for strict match i.e. 100 % match. However, when it comes to BRISK-FLANN, this rule is different. Refer to FI-9 for more info.

I. Feature Index-9

BRISK-FLANN algorithm:

This a feature and key point detection algorithm. This means, it finds key points (termed as *kp* in the tool reporting) of image features in baseline and runtime images, compares and arrives at matched good points (termed as *gp*), and the percentage of good points (termed as *gpp*).

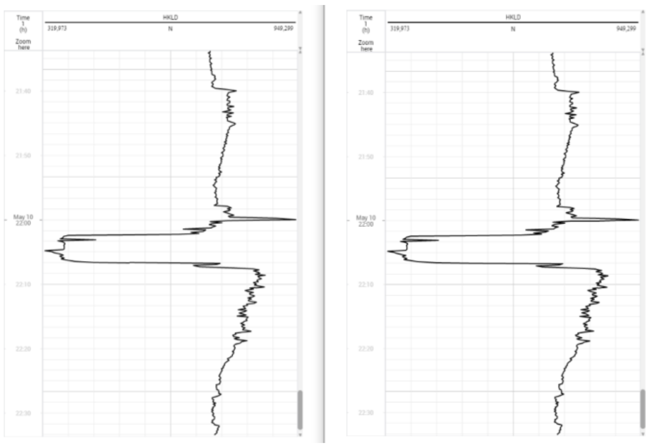
The use case of this algorithm is founded on the principles of finding key points between baseline and runtime images. If the count of *kp* is equal, images are identical. Otherwise, they are similar but not exact. Having said that, even if differences exist between the baseline and the runtime images in terms of the above described metrics, we may still be approving the differences from visual perspectives.

Hence, in the context of using this tool, we need to approve the amount of differences by subtracting the *kp* figure of runtime from the baseline. This difference figure must be specified by the users in the baseline json records. It's ok to have negative approval number too.

Where does this algo fit the bill ? In cases where there are very minor differences, but does not exist visually, and we don't want to treat such differences as discrepancies, this algo comes handy by enumerating the key points associated with the particular image feature state.

Will other algos be unable to catch such differences? Other algos too catch, but they may fail in some cases where you don't want them to fail. However, as for BRISK-FLANN is concerned – in such cases, it gives you the control at the granular level by tracking such subtleties in the form of *kps*. Having leveraged this aspect, this tool allows you to specify the allowed variance in key points between the intended images and does the comparison of this configured baseline figure with the runtime figure.

For example, details given below should give you some idea. Look at the below image – one is the baseline, and the other one is the runtime image.



And, the BRISK-FLANN metrics for the images are:

```
"confirmed_baseline_kp": "1031",
"confirmed_runtime_kp": "1069",
"confirmed_good_points": "294",
"confirmed_good_points_percent": "28.52",
"confirmed_kp_variance": "-38",
"msg": "confirmed_kp_variance - manual update needed"
```

As you can see, the difference between the two images are 38 key points. But, the image is visually still looking the same. Despite outperforming this algorithm in most of the other aspects, the other algos(except diff_hashing, but it has its flip side too) are seeing the anomalies during the matching operation of the above image set.

J. Feature Index-10

BRISK-FLANN configurations:

```
"BRISK_FLANN_bl_confirmed_variance_auto_update(disabled)":"false",
"BRISK_FLANN_parametric_baseline":"D:/Automation/DW/SDV/py/benchmark-util/Exp_BF.json",
"BRISK_FLANN_baseline_metrics_auto_update(disabled)":"false",
"BRISK_FLANN_gp_gpp_check_enabled":"true",
"FLANNmatcher_accuracy":"0.5",
```

The two configurations ending with "(disabled)" are, though implemented, will be made available after studying the practical requirements and the associated risks of losing the control.

I. *BRISK_FLANN_parametric_baseline*:

A placeholder for specifying the json file consisting of baseline records for image metrics derived by BRISK-FLANN by means of taking in baseline and runtime images.

```
{
     "image": "8.PNG",
     "confirmed_baseline_kp": "549",
     "confirmed_runtime_kp": "549",
     "confirmed_good_points": "549",
     "confirmed_good_points_percent": "100.0",
     "confirmed_kp_variance": "",
     "msg": "est.kp_vari:0.Need conf."
},
```

The configurations starting with "captured_" are meant for some feature enhancements, if need arises. It's highly recommended to have separate baseline files for each module/functionality, as the file size may grow exponentially in cases where the image count exceeds a few hundreds and may lead us to face I/O operation issues.*BRISK_FLANN_gp_gpp_check_enabled*: A flag to instruct the tool to factor in or ignore *good points* and *good points percentage* in the net result of a single image match operation. This doesn't have any effect on the *kp* match.

II. FLANNmatcher_accuracy:

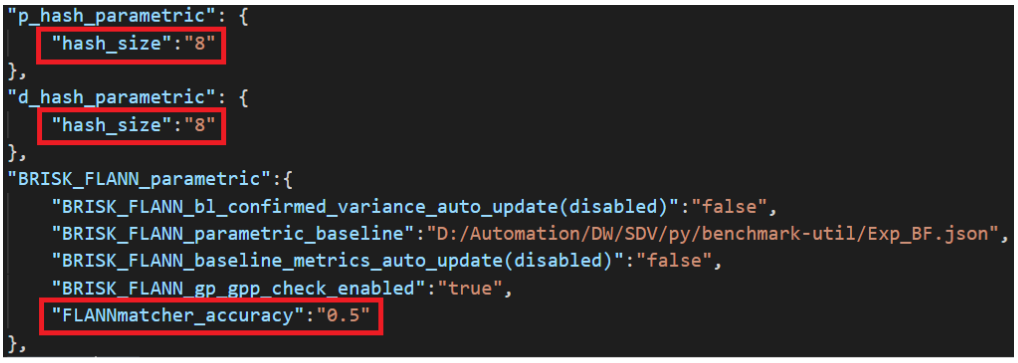
Defines the level of precision to be applied while finding matching good points

5. Evaluation Scores

Algorithm	100 % Match Score	Comments
Structural Similarity Index	1	
Perceptual Hashing	0	
Diff Hashing	0	
BRISK-FLANN	N/A	Primarily defined by accuracy config., key points count, good matching points between images, and good matching points percentage. Refer to BRISK-FLANN baseline records.

6. Evaluation Accuracy Settings

The boxed fields are signifying the accuracy level of the respective algorithm. This setting affects the evaluation scores drastically. The values shown below are optimum and can be considered for most of the evaluations. The Sampling Scores given below are derived with these accuracy settings.



```
"p_hash_parametric": {
    "hash_size":"8"
},
"d_hash_parametric": {
    "hash_size":"8"
},
"BRISK_FLANN_parametric":{
    "BRISK_FLANN_bl_confirmed_variance_auto_update(disabled)":"false",
    "BRISK_FLANN_parametric_baseline":"D:/Automation/DW/SDV/py/benchmark-util/Exp_BF.json",
    "BRISK_FLANN_baseline_metrics_auto_update(disabled)":"false",
    "BRISK_FLANN_gp_gpp_check_enabled":"true",
    "FLANNmatcher_accuracy":"0.5"
},
```

7. Sampling Scores

Image	SSI	P-Hash	D-Hash
1	0.77	20	6
2	0.74	2	0
3	0.94	0	1
4	0.94	0	1
5	0.6	12	18
6	0.63	20	8
7	0.63	20	6
8	1	0	0
9	0.91	4	3
10	0.91	2	5
11	0.99	2	0
12	0.95	4	4
13	0.99	6	1
14	0.94	0	1
15	0.99	0	0
16	0.99	0	2
17	0.97	18	1

🟨 **False Positives = you may not or don't want them to fail, but the algos have failed**

🟪 **Minor/Moderate False Negatives = you may not be ok to consider them to be not an issue. The algos have not considered the small differences**

🟥 **False Negative = you want the differences to be reported. The algo has failed to report such differences.**

- **Metrics of BRISK-FLANN are captured in BF_sampling_base1.json under BRISK_FLANN_baselines folder. It's left out in this version of the document.**

8. Performance Scores

Algorithm	Performant	Comments
Perceptual Hashing AND Diff Hashing	1	Negligible difference between p-hash and d-hash
Structural Similarity Index	2	
BRISK-FLANN	3	Multiple subsets of algorithms are at play