Twizzle - A Multi-Purpose Benchmarking Framework for Semantic Comparisons of Multimedia Object Pairs

Stephan Escher, Patrick Teufert, Robin Herrmann, and Thorsten Strufe

TU Dresden, Germany
{firstname.lastname}@tu-dresden.de

Abstract. This paper describes Twizzle Benchmarking, a framework originally developed for evaluating and comparing the performance of perceptual image hashing algorithms. There are numerous perceptual hashing approaches with different characteristics in terms of robustness and sensitivity, which also use different techniques for feature extraction and distance measurements, making comparison difficult. For this reason, we have developed Twizzle Benchmarking, which enables comparison and evaluation regardless of the algorithm, distance calculation, data set or type of data. Furthermore, Twizzle is not limited to perceptual hashing approaches, but can be used for a variety of purposes and classification problems, such as multimedia forensics, face recognition or biometric authentication.

Keywords: benchmarking · perceptual hashing · multimedia forensics · face recognition

1 Introduction

The progressive digitalization of all areas of life and the associated accumulation of large amounts of data require tools that make such amounts of data efficiently and quickly searchable, comparable and analyzable. In the multimedia area, such tools extract certain features from a multimedia object that can be used to describe the object or its content. This feature extraction is used to reduce the sheer size of multimedia objects, to prevent redundancy and noise, as well as to be stable against changes such an object undergoes during its lifetime. Working with extracted features instead of the multimedia object itself allows a faster and more efficient processing of large amounts of data. However, by reducing the information to features, the decision whether a multimedia object resembles another or if it is the content to be identified is no longer unambiguous but depends on probabilities or thresholds. Thus, the quality of these tools is measured on the one hand by the robustness of the extracted features to changes of the multimedia object and on the other hand by their sensitivity to different multimedia objects.
An example to illustrate could be perceptual image hashing (PIH), an umbrella

term for hash functions which produces a short and fix-sized fingerprint out of an image file, based on the perceptible content, e.g. the structure of the scene. The goal of these PIH algorithms is to find image duplicates while being robust to perceptual preserving transformations (PPT), like rotation, compression or brightness adjustments, as well as sensitive to similar images.
The workflow of a PIH algorithm usually starts with preprocessing, like scaling [14], ring partition [11] or segmentation [8]. Subsequently, perceptual features are extracted from the preprocessed image. Examples are methods based on Wavelet coefficients [12], Fourier-Mellin coefficients [9] or Local Binary Patterns [2]. Finally the extracted features are often quantized and represented as a vector of fixed length. In order to examine two images for perceptible similarity, their generated hashes are subjected to a distance comparison. Fields of application range from reverse image search, image authentication [10] and digital forensics [8] up to phishing website detection [4].
So far, there are many different PIH approaches. Each of these algorithms allows different PPTs, but often solve only one aspect such as an affine transformation with high performance. The evaluation of these algorithms vary widely, usually considers only a partial area of robustness and are only compared with algorithms of similar design. In addition, combinations of transformations and the sensitivity characteristics are rarely taken into account. Existing Benchmarking solutions which try to solve these issues are PHabs [16] and Rihamark [15]. However, for Phabs today neither code nor an exact description of the structure can be found and is therefore outdated. The developers of Rihamark have already critically noted this and hence presented their own framework. Rihamark provides a plugin system and comes with predefined PPTs and PIH algorithms as well as support for sensitivity evaluation. Nevertheless Rihamark is limited to PIH algorithms that generate a binary hash and compare them with hamming distances, which means that algorithms with a different structure cannot be compared. Furthermore its dependent on the data set (only images), difficult to extend, difficult to apply to large data sets and to deploy on servers with higher resources.
For those reasons we propose our modern and modular benchmarking framework Twizzle, with which existing and new PIH algorithms can be easily evaluated and compared, by remedying the weaknesses of existing benchmarks. Due to the modular design and the independence of feature extraction, decision making and the data set, twizzle is not limited to PIH approaches, but can be used for a variety of similar applications.

2 Twizzle Benchmarking

Facing that every PIH algorithm has its own way of hash representation and distance calculation we realized that we have to abstract the task to the following Question: Are two objects (in this case images) the same or not? Therefore we have designed a benchmarking pipeline in which the algorithms to be compared have to solve this task independently of their feature extraction and decision

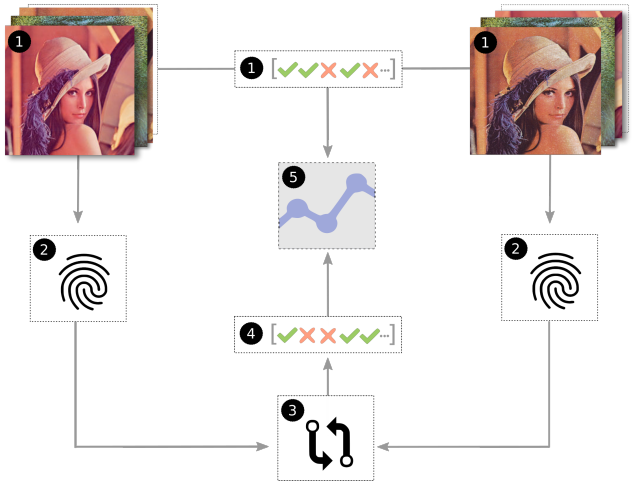

Fig. 1. General workflow of Twizzle. (1) Input: Challenge - objects to compare and ground truth (2) feature extraction (3) decision making (4) Output: list of decisions (5) Analysis: decisions vs. ground truth

making. The resulting framework is written in python 3 and freely available[1] under GPLv3. The whole workflow of the benchmarking pipeline could be seen in Figure 1.
Overall, Twizzle consists of a "Challenge creator", the algorithm tests and the "Analyser", where the algorithm tests can be further separated into "Wrapping" the algorithms to be evaluated and the "Test runner". Each of these parts can be independently reused in several different pipelines for different algorithms and different problem cases.

2.1 Challenge Creation

The first step of the benchmarking pipeline represents the creation of a specific challenge an algorithm has to solve (see Fig. 1 step 1). Twizzle originally was designed for challenges which have the form of a pairwise comparison of multimedia objects. I.e. the algorithm has to decide whether the objects or the content within the objects are the same or not. Thus a data set with comparison pairs, each consisting of an original object paired with a comparative object, has to be created. For each pair the expected decision (ground truth) needs to be specified. I.e. if a pair consists of the same objects the decision should be "True" ("These are the same objects.") otherwise "False".
For PIH-Challenges, object pairs could for example consist of similar but not same images for a sensitivity challenge, of same but modified images for a robustness challenge or of both as a practical use case challenge (see List 1.1).

[1] github.com/dfd-tud/twizzle

From a practical point of view, the first step is to initiate a new instance of Twizzle where a name for the database, in which challenges and results will be saved, needs to be specified. Furthermore, a list of paths to original objects, a list of corresponding comparative objects and a Boolean list of ground truth decisions for each original-comparative pair has to be created. Dimensions of the lists of original objects, corresponding objects and ground truth decisions need to be the same. Optionally, additional metadata can be passed as python dictionary, for example to describe the specific challenge. Finally the created challenge is added to the database. Thus the challenge is prepared for any number of tests, such as the evaluation of different algorithms or the testing of different parameters of an algorithm.

Listing 1.1. Example Challenge Creation

```
DBPath = "test.db"
tw = Twizzle(DBPath)
sChallengeName = "pih_challenge"
aOriginals = ["img1.png", "img1.jpg", "img2.png"]
aComparatives = ["img1_r10.png", "img1_r20.png", "img1.png"]
aTargetDecisions = [True, True, False]
dicMetadata = { "transform": "rotation" }
tw.add_challenge(sChallengeName, aOriginals, aComparatives, aTargetDecisions, dicMetadata)
```

2.2 Wrapping an Algorithm

Next a wrapper for each algorithm to be evaluated has to be created. Wrappers need to have at least two input parameters for the original and comparative objects, which are specified in the created challenge. Additional named arguments can be passed, like different parameters for the algorithm to be evaluated. The first step of such a wrapper function is to load the objects linked to in the two lists. Then, for each object pair, it has to evaluate whether the two objects are the same or not, which is done via the user-defined algorithm.
An exemplary structure of a wrapper for a user-defined PIH-algorithm could consist of iterating over each image pair, extracting the features and generating the perceptual hash for each image (see Fig. 1 step 2). With the user-defined decision making algorithm, e.g. based on a normalized hamming distance of the two hashes and a threshold, it is decided if both hashes represent the same image or not (see Fig. 1 step 3). Finally, the wrapper must return the list of algorithm decisions for all object pairs (see Fig. 1 step 4) and if desired additional arbitrary metadata.

Listing 1.2. Example Wrapper

```
def wrapper(aOriginalImages, aComparativeImages, param1, ...)
    for i, sOriginalImage in enumerate(aOriginalImages):
        sComparativeImage = aComparativeImages[i]
        hashOriginal = algorithm(sOriginalImage, param1)
        hashComparative = algorithm(sComparativeImage, param1)
        deviation = distance(hashOriginal, hashComparative)
        if(deviation <= threshold): bDecision = true
        aDecisions.append(bDecision)
    return(aDecisions, dictMetadata)
```

2.3 Test Runs

Tests for Twizzle are black box tests. This means, that the internal workings of an algorithm are not known to Twizzle. All Twizzle expects from the user-defined algorithm is that it can handle a set of original objects and corresponding comparative objects and return some kind of decision values, which is done through the wrapper.
"Test runs" specifies which algorithm has to solve which challenge and provide any additional parameters for "Test wrappers". All decisions made during a test execution are returned to the Twizzle framework, where it compares the algorithm decisions with the ground truth decisions specified during "Challenge creation" and calculates the TPR, TNR, FPR, FNR, accuracy, precision and F1 score (see Fig. 1 step 5). Additional also user-defined metadata can be returned by each test, for example the used algorithm parameters. Based on this outputs an algorithm can be easily compared to others. Tests defined in Twizzle are executed in parallel with a user-defined number of threads and can therefore also be set up on a cluster.

Listing 1.3. Example Test Run

```
oRunner.run_test_async("pih_challenge", wrapper, {"param1":param1})
```

2.4 Analyse Results

Twizzle also provides an Analysis component, which will collect and merge all tests and the corresponding challenges and returns a pandas dataframe [6]. This dataframe contains the test results, evaluation metrics per test and all metadata added during "Challenge creation" and running the actual test. Comparing tested algorithms can be easily done due to Twizzle abstracting the binary classification task and generating typical classification evaluation metrics. The evaluation metrics provided by Twizzle include custom metrics, challenge name as well as the metrics mentioned above.

2.5 Twizzle Features

Overall Twizzle accepts any user-defined feature extraction and decision making algorithms, independent of their type and functionality and enables the comparison of them. Further, Twizzle enables the creation of user-defined Challenges, consisting of user-defined data sets independent of their data type, with which the algorithms can be evaluated. Although Twizzle was developed for image comparisons, it can therefore be used for any pairwise comparison task, e.g. of text documents. Thereby Tests and Challenges are independent of each other, i.e. each created Challenge can be used for other Tests, as well as each created algorithm wrapper can be tested for any challenge. Twizzle represents the test data and evaluation metrics for analysis of each test and simplifies the comparison of the results. Further tests and analysis can be extended with user-defined meta data. Finally test runs could be executed easily in parallel.

3 Use Cases

Twizzle's scope is not limited to perceptual hashing approaches, but can be used for a variety of purposes. This section contains some examples that describe the applicability of Twizzle to different use cases. Other use cases which are not mentioned here could be for example object detection or biometric authentication.

3.1 Multimedia Forensics

From a forensic point of view it is important to be able to determine the source device (e.g. a printer) of an unknown multimedia file (e.g. print-out) or at least to compare multimedia data with respect to their source device, e.g. for the analysis of blackmail letters. Forensic methods try to solve this problem by extracting so-called intrinsic signatures out of the multimedia data. These are artifacts caused by the source device during the creation of the multimedia file. Depending on the stability and distinguishability of the intrinsic signature, it can then be used as a fingerprint for the corresponding source device, device model or type.
In printer forensics for example intrinsic signatures like geometrical distortions of text [13] and image [1] elements, the texture and structure of printed characters [3] or the halftoning structure [5] could be used to identify the source printer model. All of these fingerprints have different properties regarding robustness (influences such as type of paper) and sensitivity (discriminating a bunch of printers). Furthermore their exist a lot of various algorithms which try to extract these signatures in different ways and even decide with different decision making algorithms (e.g. classification, euclidian distance or correlation with a reference pattern). Just like the perceptual hashing approaches, all of this methods use their own curtailed evaluations concerning sensitivity, robustness and data sets. However such different signatures, extraction algorithms, their settings as well as the combination of such methods could easily be evaluated and compared with Twizzle.
For the challenge of comparing two print-outs based on their signature similarity, the benchmarking pipeline looks exactly like for a PIH approach. Therefore the data set could consists of scanned print-outs from same and different printer devices with different robustness parameters (like different fonts, print settings or paper types), like:

Table 1. Example challenge for a printer forensic algorithm

original	[”printer1_font2.png”, ”printer2.jpg”]
comparative	[”printer1_font1.jpg”, ”printer1.png”]
ground truth	[True, False]

After extracting the fingerprint of original and comparative print-out, the decision algorithm used decides whether they are from the same source device

or not. After the test runs, decisions and ground truth are compared whereby tested algorithms can be compared and evaluated, independent of used signatures, extraction algorithm or decision making. For a source printer classification challenge, Twizzle could also be used while be prepared as described in the next use case.

3.2 Face Recognition

Another use case could be face recognition. Face recognition is the task of identifying the person depicted in an image. Various biometric features (facial features) are extracted from the images, which often vary for different algorithms. These face features should ideally be robust to changes in image characteristics (brightness, contrast, resolution, aspect ratio etc.) and physical changes (beard, glasses, pose, hairstyle, scars, etc.) [7, 17]. At the same time they should be sensitive, i.e. characteristics of different persons should be distinguishable in any case. We want to give an overview of how Twizzle can be used for the classification problem "Two pictures are given, is the same person shown?".

Challenge Creation For a data set with face images and corresponding personal labels, the creation of challenges can be done as follows. A "Challenge creation" for a classification problem, e.g. using machine learning, would be done by splitting the data set into train and test data and then training the machine learning model on the train data. The test data can then be used to generate original-comparison object pairs, and the specification of the ground truth using already labeled images can be done simply by comparing the generated image pairs and the corresponding labels. If for a pair of images both images have the same label, the algorithm to be tested should return true, otherwise false. The generated pairs and the corresponding ground truths are stored for this challenge.

Test Wrapper The test wrapping for our scenario can be done by generating labels for each image pair and comparing these labels. For each image in a pair, the label is predicted using the trained model. The decision is made by comparing the predicted image labels and adding the decision to the total decisions made, which the test returns at the end.

Test Run The test runner itself simply specifies which "Test Wrapper" is used and in a machine learning scenario provides the test with the trained model as additional parameter, since in this case the model is needed by the algorithm to be tested.

Analysis The analysis is individually necessary and is determined by the self-defined requirements for a "good" algorithm. When developing a face verification system for security, a "good" algorithm may need to have a very low false positive rate, while a higher false negative rate is acceptable. On the other hand, an image retrieval task could be the opposite case, where a "good" algorithm finds all images for a person and delivers false positive results rather than missing an image.

4 Conclusion

We have developed Twizzle, a multi-purpose benchmarking framework for various comparison tasks, originally designed to compare algorithms that determine whether multimedia objects are the same or not. Twizzle provides an easily extensible architecture to run and analyze many tests in parallel and for different problem scenarios. Twizzle users are provided with many important evaluation metrics for each test, making it easy to compare different algorithms for the same task and optimize algorithms for specific score metrics. Due to the high reusability of the Twizzle components, the algorithms can be quickly evaluated on multiple data sets by changing only the underlying challenge without having to develop a completely new pipelining and testing process.

References

1. Bulan, O., Mao, J., Sharma, G.: Geometric distortion signatures for printer identification (2009)
2. Davarzani, R., Mozaffari, S., Yaghmaie, K.: Perceptual image hashing using center-symmetric local binary patterns (2016)
3. Ferreira, A., Navarro, L.C., Pinheiro, G., Santos, J.A.d., Rocha, A.: Laser printer attribution: Exploring new features and beyond (2015)
4. Joshua S., W., Jeanna N., M., John L., S.: A method for the automated detection phishing websites through both site characteristics and image analysis (2012)
5. Kim, D.G., Lee, H.K.: Colour laser printer identification using halftone texture fingerprint (2015)
6. McKinney, W.: Data structures for statistical computing in python (2010)
7. Sharif, M., Naz, F., Yasmin, M., Shahid, M.A., Rehman, A.: Face recognition: A survey. (2017)
8. Steinebach, M., Liu, H., Yannikos, Y.: Efficient cropping-resistant robust image hashing (2014)
9. Swaminathan, A., Mao, Y., Wu, M.: Robust and secure image hashing (2006)
10. Tabatabaei, S.A.H., Ur-Rehman, O., Zivic, N., Ruland, C.: Secure and robust two-phase image authentication (2015)
11. Tang, Z., Zhang, X., Li, X., Zhang, S.: Robust image hashing with ring partition and invariant vector distance (2016)
12. Venkatesan, R., Koon, S.M., Jakubowski, M.H., Moulin, P.: Robust image hashing (2000)
13. Wu, Y., Kong, X., You, X., Guo, Y.: Printer forensics based on page document's geometric distortion. (2009)
14. Yang, B., Gu, F., Niu, X.: Block mean value based image perceptual hashing (2006)
15. Zauner, C., Steinebach, M., Hermann, E.: Rihamark: perceptual image hash benchmarking (2011)
16. Zhang, H., Schmucker, M., Niu, X.: The Design and Application of PHABS: A Novel Benchmark Platform for Perceptual Hashing Algorithms (2007)
17. Zhao, W., Chellappa, R., Phillips, P.J., Rosenfeld, A.: Face recognition: A literature survey (2003)