Author's accepted pre-print
DFRWS EU 2024

PHASER: Perceptual Hashing Algorithms Evaluation and Results - an Open Source Forensic Framework

Sean McKeown[a], Peter Aaby[a], Andreas Steyven[a]

[a]*School of Computing, Engineering, and the Built Environment, Edinburgh Napier University, Edinburgh, UK*

Abstract

The automated comparison of visual content is a contemporary solution to scale the detection of illegal media and extremist material, both for detection on individual devices and in the cloud. However, the problem is difficult, and perceptual similarity algorithms often have weaknesses and anomalous edge cases that may not be clearly documented. Additionally, it is a complex task to perform an evaluation of such tools in order to best utilise them. To address this, we present `PHASER`, a still-image perceptual hashing framework enabling forensics specialists and scientists to conduct experiments on bespoke datasets for their individual deployment scenarios. The framework utilises a modular approach, allowing users to specify and define a perceptual hash/image transform/ distance metric triplet, which can be explored to better understand their behaviour and interactions. `PHASER` is open-source and we demonstrate its utility via case studies which briefly explore setting an appropriate dataset size and the potential to optimise the performance of existing algorithms by utilising learned weight vectors for comparing hashes.

Keywords: Evaluation Framework, Perceptual Hashing, Hashing, Content Matching, Image Forensics

1. Introduction

Much like other cyber security disciplines, the field of digital forensics is faced with constant external pressure to adapt to new developments, whether they are societal or technological. Despite this rapid change, digital artefacts need to adhere to solid scientific principles in order to be robust enough to rely on in court, as this is the fundamental purpose of the discipline. To this end, there have been initiatives to perform robust testing of forensics tools [1], open-sourcing standardised datasets to facilitate experimental testing and tool evaluation [2], and more recently to formalise the process of forensic experimentation itself [3]. However, there still appears to be a disconnect between digital forensics as a practice and as a science [4], with a reliance on the output of tools without an accompanying robust understanding of their accuracy or stability [5].

To compound the problem, Law Enforcement Agency (LEA) caseloads and evidence volumes have been a problem for nearly two decades [6], unavoidably placing further emphasis on automation [7]. As such, evaluating and understanding tooling has never been more important, particularly as the field moves towards further use of Artificial Intelligence technologies, which require large datasets to properly evaluate [8].

One such use case for automation in digital forensics is that of identifying known files, which traditionally was served by cryptographic hashing for exact binary matching [9], but now also constitutes approximate, similarity-based, matching schemes [10]. The performance of such algorithms is important, particularly in use cases such as the detection of Child Sexual Abuse Material

Email addresses: `S.McKeown@napier.ac.uk` (Sean McKeown), `P.Aaby@napier.ac.uk` (Peter Aaby), `A.Steyven@napier.ac.uk` (Andreas Steyven)

(CSAM), the scale of which has been increasing in recent years [11]. Perceptual (similarity) Hashing has been deployed on the cloud, largely via Microsoft's PhotoDNA [12], for some time, with LEA tools such as Magnet AXIOM offering plugin support. At the same time, Facebook [13] and Apple [14] also have their solutions. However, despite much interest in the area, the field lacks a common evaluation framework for perceptual hashing, despite their being counterparts for binary-based approximate matching [15, 16].

To address this gap and to further facilitate a culture of scientific evaluation for automated forensics artefact processing, we introduce `PHASER` (Perceptual Hashing Algorithms Evaluation and Results), a perceptual hashing evaluation framework. Inspired by `FRASHER` [16], an evaluation framework for binary similarity hashing, `PHASER` is built on modern scientific Python libraries (e.g. Pandas, SciPy), with a modular approach to facilitate various evaluation conditions.

In the remainder of this paper, we describe the background, related work and problem space in Section 2, followed by a description of `PHASER` and its use cases in Section 3. We then present short indicative experiments using the framework in Section 4, with conclusions in Section 5.

2. Background and Related Work

2.1. Content-Based Image Retrieval and Forensic perceptual hashing

The field of Content-Based Image Retrieval (CBIR) is the progenitor of perceptual hashing, tracing its extensive body of work back as early as the 1970s [17]. The problem of retrieving similar images for a given query is a difficult one, and new methods are still being developed and benchmarked/evaluated in dedicated conferences, such as ACM's ICMR. The field is mature, and there is a clear understanding of the pipeline for preprocessing, feature extraction, image representation, and similarity matching. Perceptual hashing, particularly in the digital forensics context, is less well understood [18], particularly as there are additional constraints on the process.

The general task in CBIR is to retrieve a list of relevant images, such as images of trees for a query picture of a tree. Metrics used to evaluate the performance of these systems, such as Precision, Recall, and the F1-measure, reflect the retrieval of a group of *relevant* images for a query image, though this idea of relevance may be somewhat fluid and difficult to capture [19]. Modern systems may also use multiple indices of features, as well as feedback loops, to refine results [17].

The retrieval task in a digital forensics context is quite different, however, and rather than a loose sense of topic or object relevance the scenario is largely the retrieval of duplicates, or near-duplicates, of an image [10].This shifts the evaluation criteria from one based on relevance to a biometric style evaluation, where True/False Positives/Negatives are assessed in a confusion matrix. Essentially, perceptual hashes are treated as a natural extension of traditional cryptographic good/bad lists, looking for 'hits', with the caveat that non-exact hash matching necessitates some similarity metric and corresponding threshold to consider an observation a 'hit' or match.

However, this decision threshold introduces a tension between the ability to discriminate between unrelated images and robustness to attacks or modified images [10], as measured by inter- and intra-image analysis [20], respectively. If the threshold for a 'hit' is set too loosely, images which are unrelated may be matched, creating additional false positives. Conversely, setting the threshold too restrictively will miss authentic matches, introducing false negatives. In a forensics use case, to avoid overwhelming a forensic practitioner at scale, prior work has suggested that a false positive rate of 1% is excessive, but that it may be acceptable to have a false negative rate of 1% [21]. However, a lack of understanding of these trade-offs, or how to achieve them, is not ideal for practitioners when utilising tools [22].

While comparisons between feature vectors in CBIR can use a wide number of distance metrics [17], perceptual hashes are almost exclusively compared using the normalised Hamming distance between two hashes. As with traditional cryptographic hashing, and particularly impor-

tant in the CSAM use case, perceptual hashes are intended to be irreversible and not leak information about the image content in the hash[1], such that more direct feature comparisons are eschewed in favour of these criteria. While this does limit the possibilities, there is arguably some room for innovation here, particularly as Hamming distance is essentially a global measure and does not capture the locality of bit differences, which may lose useful information. Similarly, algorithmic properties may mean that not all bits contribute equally when matching images, with bit weight optimisation being explored in prior work [21, 26].

2.2. Evaluating perceptual hashing

Aside from the widespread use of a shared distance metric (Hamming distance), approaches to evaluating perceptual hashes are much more varied and inconsistent than those used in CBIR. Exploratory distance-based analyses have been used to understand the distributions of distance scores [10, 27, 28, 29]. Threshold analysis can then be conducted to establish a distance cut-off, used to decide what constitutes a match, at which point CBIR evaluation metrics (Precision, Recall, F1, Precision@n) are often used [27, 30, 31, 32, 33]. Other approaches make use of raw False Acceptance/Rejection rates and confusion matrices [10, 20, 34], with Receiver Operating Characteristics (ROC) curves sometimes being used to visualise these trade-offs across the spectrum of possible thresholds [20, 35]. There does not appear to be a consensus on dataset size or number of pairwise observations, with test datasets ranging from tens [10, 20], to approximately one-thousand [27, 36], and tens of thousands or more [28, 29, 31, 33].

There have been works that seek to create a uniform framework for assessing perceptual hashes. However, the most recent we could find is designed for reverse image search [31], and Zauner's original framework, Rihamark (2010) [20] did not appear completely functional when we tested it. As such, it seems prudent to provide the community with a modern toolkit to facilitate evaluation with a low barrier to entry.

[1]Though this is not necessarily the case in practice [23, 24, 25].

3. PHASER

3.1. Design Goals

There were several main aims when designing `PHASER`: **i)** It should be open-source with modular components. In this case, the modular elements correspond to the Perceptual Hashing Algorithm, Image Transform (manipulation), and Distance Algorithm. **ii)** It should be plug-and-play with a provided dataset with minimal configuration, allowing experienced developers to build on it. **iii)** Generated hashes and distance scores should be portable, allowing for external analysis.

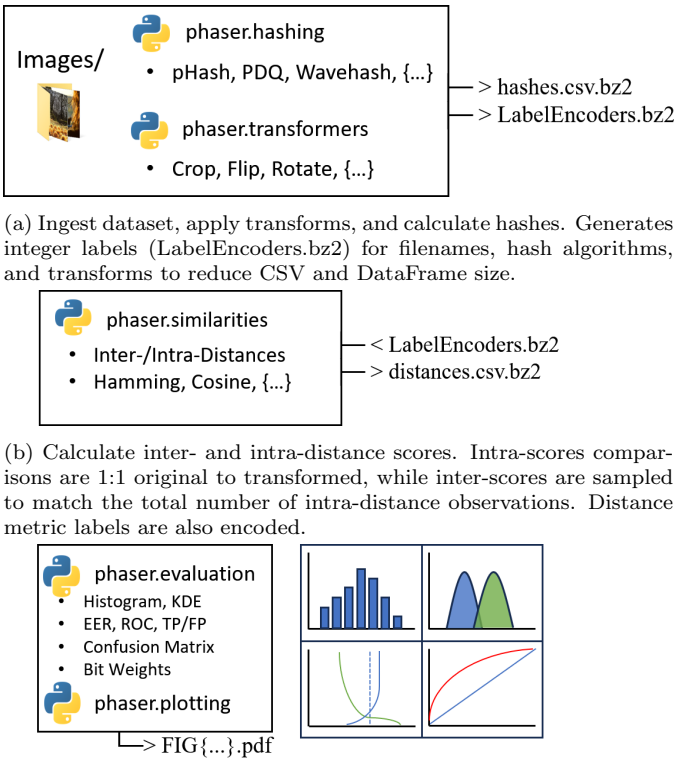
We achieve i) by providing separate modules to define each element, making it straightforward to import additional algorithms or provide a function to implement a new approach. For ii), we provide a Jupyter notebook to facilitate processing a specified dataset and simplify the algorithm selection and analysis stages. For iii), we ensure that intermediate hashes and distance scores are saved to disk in CSV format while enabling transformed images to be saved to disk (with temporary memory-only storage by default).

3.2. Overview and Processing Pipeline

Here, we provide an overview of the layout and processing steps in `PHASER`. We choose to explicitly separate the analysis of inter-distances, which assume image content independence, and intra-distances, which are measured between original and transformed images, as with Zauner's original framework [20]. `PHASER` is implemented in Python 3, and efforts have been made to leverage the power of modern data science libraries, such as NumPy, SciPy, and SciKit-Learn. `PHASER` is an open-source contribution and is available on GitHub[2].

```
phaser
├── hashing/
│   ├── _algorithms.py
│   │   ├── class PerceptualHash(ABC) // Abstract class
│   │   ├── class PHash(PerceptualHash) // Wrap pHash in class
│   │   └── class {...}
│   └── _helpers.py
│       ├── class ComputeHash
│       │   // Joblib wrapper for sim_hashing, return DataFrame
│       └── def sim_hashing(img_path, transforms, algorithms)
├── similarities/
│   ├── _distances.py
│   │   ├── def my_metric // Calculate hash vector distance
│   │   └── def {...}
│   └── _helpers.py
│       ├── class IntraDistance
│       │   // Wrap scipy.spatial.distance.cdist
│       └── class InterDistance
│           // Wrap scipy.spatial.distance.pdist
├── transformers/
│   └── _transforms.py
│       ├── class Transformer(ABC) // Abstract Class
│       ├── class Rotate(Transformer) // Return PIL object
│       └── class {...}
├── evaluation/
│   └── _evaluation.py
│       ├── class MetricMaker
│       │   // Confusion Matrix, ROC, FPR/Distance analysis
│       └── class ComputeMetrics // Parallelise MetricMaker
├── plotting/
│   └── _plotting.py
│       // Histogram/KDE plot, Confusion Matrix/EER/ROC axes
└── utils/
    └── _utils.py
        // Hash formatting, image loading, label encoding/decoding
```

Figure 1: Tree Structure of the PHASER Library



(a) Ingest dataset, apply transforms, and calculate hashes. Generates integer labels (LabelEncoders.bz2) for filenames, hash algorithms, and transforms to reduce CSV and DataFrame size.

(b) Calculate inter- and intra-distance scores. Intra-scores comparisons are 1:1 original to transformed, while inter-scores are sampled to match the total number of intra-distance observations. Distance metric labels are also encoded.

(c) Calculate statistics and plots. Distance distribution histograms/KDE plots, EER analysis for distance thresholds, ROC curves and confusion matrices are provided as a baseline.

Figure 2: Stages of processing in the PHASER Library

3.2.1. Step 1 - Hashing and Transformation

The tree structure of the library is depicted in Figure 1, with processing stages in Figure 2. In **Step 1** (2a) the specified folder of original images is ingested by ComputeHash, which wraps calls to Python's Joblib to parallelise hashing and transformation. The process is roughly: *i)* Load each image into an object from the Python PIL library. *ii)* Generate hash digests for each specified algorithm, with definitions located in phaser.hashing. *iii)* Generate each image transform (with definitions in phaser.transformers), deriving hash digests in the same way as the original images. To conserve RAM and disk space, the DataFrame, which stores hashes, encodes filenames, algorithm names, and transforms names into integers (LabelEncoders.bzip2 in Figure 2). Hashes can also be generated externally and imported into Stage 2 with an appropriately formatted CSV file.

As noted previously, transforms are handled in-memory, but can optionally be written to persistent media. Additionally, transforms can be loaded *from* the disk if they already exist. This is particularly useful in cases where they cannot be constructed easily in the framework (for example, importing thumbnails generated by Operating System caches [32]). For existing perceptual hashes, registering them in PHASER involves writing a wrapper class to format the hash digest as a string of bits, formatting which is facilitated by phaser.utils. Transforms also need to extend the base Transformer class and return a PIL object but can generate the image via any means. In both cases, classes must be exported outside the module by updating the corresponding module's __init__.py file.

We do not formally benchmark the computational performance of PHASER. However, anecdotally, a Ryzen 5900X test system, reading the Flickr 1 million [37] dataset from a SATA SSD, extracts hashes at ≈155 files per second (<2 hours) for three hashes and five transforms.

[2]DOI: https://zenodo.org/doi/10.5281/zenodo.10363150. GitHub: https://GitHub.com/AabyWan/PHASER

3.2.2. Step 2 - Pairwise Distance Calculations

The labels and DataFrame/CSV of hashes from Step 1 are re-used or loaded from the disk, prior to calculating pairwise distances in **Step 2** (2b). Distance metrics can be defined in phaser.similarities, and exported in a similar manner to transforms and hashing algorithms. The main difference here is that algorithms included in scipy.spatial.distance can be included by simply specifying the string associated with the algorithm, avoiding the need to pass a reference to the function directly. This is because inter- and intra-distances are calculated using SciPy's `pdist` and `cdist` functions, respectively, which facilitates easy re-use of its built-in distance metrics. Built-in metrics also support a weights vector argument, which will be relevant to our discussion in Section 4.2.

`PHASER` explicitly splits out inter- and intra-distance classes, which are marked in the DataFrame as class 0 and 1, respectively. Inter-distances correspond to the behaviour of an algorithm/transform/distance metric triplet when considering unrelated images, while Intra-distances correspond to images that should match. These intra-distances are calculated on a 1:1 basis between the original image and the hash algorithm/transform/distance triplet, meaning that for a dataset of 1,000 images, with two perceptual hash algorithms, five transforms, and two distance metrics, the overall number of intra-distance observations would be $1,000 \times 2 \times 5 \times 2 = 20,000$.

As inter-distance pairwise comparisons scale with the square of the number of images, inter-score observations are sampled, such that each image is compared with n randomly selected images in the DataFrame for each triplet. n is automatically calculated to balance both classes, and while this balancing is usually inexact it can be compensated for in the evaluation stage. This equity in scale allows for metrics and plots to be meaningfully compared on the same axes (otherwise, probability density distributions become unbalanced, for example). The DataFrame for all pairwise observations is saved to a bz2 compressed csv, with metric label encodings being present in the existing encoding file, or added at this stage.

It should be noted that while the underlying metrics are often measuring the *distance* between hashes, some evaluation approaches are preferable to be viewed in terms of *similarity*. As both values are normalised between 0 and 1, the point of view is inverted by simply calculating $1 - x$, where x is either similarity or distance.

3.2.3. Step 3 - Evaluation and Plotting

As discussed in Section 2.2, there are several ways to evaluate the behaviour of perceptual hashing algorithms. We assume that the matching task in digital forensics is essentially a classification task for some decision threshold t, with an emphasis on associated metrics such as FP/FN assessment, which is in line with Rihamark [20]. `PHASER` facilitates three types of evaluation: *i)* Exploratory Analysis, *ii)* Classification Efficacy, and *iii)* Performance Optimisation. Each is discussed in turn below.

Exploratory Analysis: Gaining an initial understanding of the behaviour of a hashing algorithm/transform/distance triplet can be a useful beginning. For this reason, we provide features to calculate and plot distance/similarity histograms for inter- and intra-comparisons, as in prior work [27, 28]. Kernel Density Estimation (KDE) plots can also be used to visualise the separation or overlap of these inter- and intra-distance behaviours. We expand on this idea in Section 3.3.

Classification Efficacy: KDE plots help understand the general trade-offs; however, actual classification performance can be viewed at the macro-level, across all thresholds, with a ROC (Receiver Operating Characteristic) curve and corresponding Area Under the Curve (AUC), or at the micro-level for individual thresholds with a Confusion Matrix. We also facilitate Equal Error Rate (EER) plots to visualise the trade-offs differently from the KDE plot. However, we don't expect that the EER is particularly useful in and of itself in a forensics use case, as it is more likely that a false negative rate would be calculated for a target false positive rate.

Performance Optimisation: When conduct-

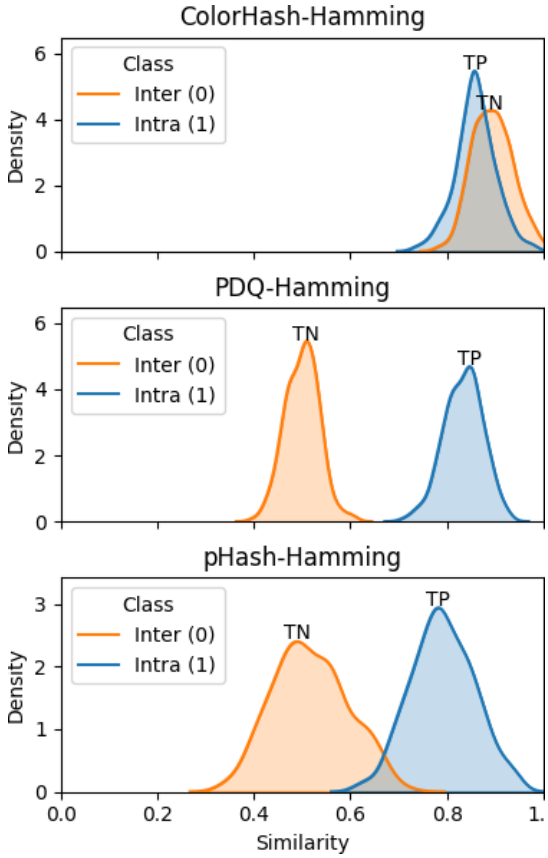
ing the above evaluation steps, there is an option to do a bit-level analysis of hash vectors to understand if patterns are generated when measuring distance scores for a given algorithm and distance pair across transform types. This data can then be used to 'learn' weights for each bit in the hash corresponding to how effective it is across all transform classes, ideally optimising performance in a similar vein to Steinebach [38]. This requires that the distance metric accepts a weight vector parameter equal in length to the chosen hash length. In SciPy, the weights are passed in as the w parameter. This step is optional, and we experimentally explore its potential in Section 4.2.

In terms of implementation, as depicted in Figure 1, evaluation statistics/metrics are separated from the plotting aspect. The `MetricMaker` class generates the statistics described above for a given triplet, with the `ComputeMetrics` class acting as a wrapper class for doing this across all triplets. Adding new statistical derivations to the `MetricMaker.fit` function would allow them to be easily processed and aggregated. The `BitAnalyzer` class can optionally generate the bit-analysis and optimisation weightings described above. The Plotting module provides wrappers to relevant Python libraries to easily visualise triplet information in a Histogram, KDE, EER, or ROC plot or as a Confusion Matrix.

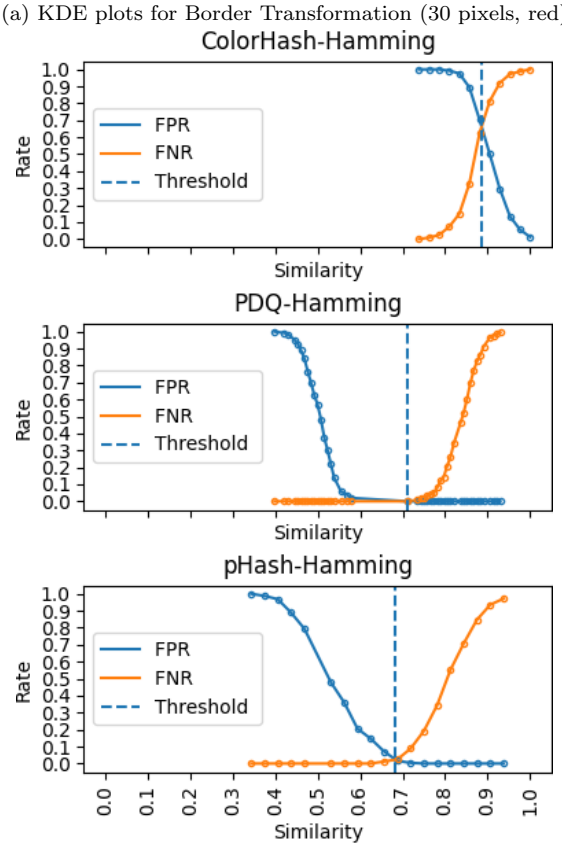
To verify that the results produced by `PHASER` were correct, we ran the same dataset as prior work [28] , using TransformFromDisk to load existing transforms, and compared statistics for perceptual hashes in Python's ImageHash library. No code base was shared, however, results were identical.

3.3. Visualising Performance - An Example

To demonstrate typical performance classes and aid in interpreting the visualisations generated by `PHASER`, Figure 3 depicts a collection of KDE and EER plots for ColorHash [39], PDQ [13], and pHash [39]. Plots were generated from a small test set for the Border transform (red border around image, width 30 pixels) as it neatly separates out behaviour types.



(a) KDE plots for Border Transformation (30 pixels, red)



(b) EER plots for Border Transformation (30 pixels, red)

Figure 3: Comparison KDE and EER plots for different algorithms on the Border transform for a small test set.

In Figure 3a, ColorHash does not separate the similarity distributions of unrelated images, the inter-class, from those that should match intra-class, when comparing originals to transforms. The overlap is considerable, such that any similarity threshold would result in many false positives, with the rate of this trade-off being visualised in the EER plot in Figure 3b. In this case, setting a threshold of 0.9 to balance the False Positive Rate (FPR) and False Negative Rate (FNR) rate results in both being around 70%, which is very poor. PDQ, on the other hand, neatly separates out both distributions, such that a suitable threshold can be found while minimising both FPR and FNR. For pHash, the distributions overlap somewhat, such that a trade-off must be made between a desired FPR and FNR rate for any threshold. While this example is specific, it is indicative of the three types of behaviour which may be present for a given algorithm/transform/distance metric triplet.

3.4. Suggested Avenues of Exploration in Perceptual Hashing

As `PHASER` facilitates the exploration of algorithm/transform/distance triplets, it should now be much more straightforward to identify algorithmic weaknesses. Identifying such issues then allows for the development of approaches to making the hash more robust without fundamentally altering the underlying mechanisms. This could be achieved in a variety of ways, such as by exploring pre-processing normalisation approaches [38]. Similarly, as is more common in CBIR, multi-index approaches, with multiple hashes, may be used to compensate for the weaknesses of each algorithm or to form a pipeline, such as those used in spam detection [40]. Building on this, a more nuanced understanding of global vs. local features, as in CBIR, would be beneficial. For example, in testing `PHASER` we noted that difficult transforms such as Rotate [10, 38] and Mirroring [10, 28] had no detrimental effect on ColorHash, as its global colour histograms are unaffected by the structure of these pixels, while other algorithms struggle greatly with x-axis mirroring.

The distance metric space also does not appear to have been adequately explored. CBIR neatly separates the ideas of representation (feature vectors, hash) and comparisons, while the comparison of features has considerably less attention paid to it in the perceptual hashing literature. Exploring different distance metrics, metric weightings, or combinations of distance metrics, perhaps together with multi-hash indices, may be beneficial. Indeed, we explore one aspect of this in the next section. Understanding structural aspects [41] of hashes generated by perceptual hashing algorithms may also be useful, as, beyond weightings, there may be recurring patterns or information leakage [23]. Such insights may also be of use when considering hash comparisons and global vs. local changes in the hash. For instance, Hamming distance is a global difference measure, but perhaps it would be worth exploring if local, clustered, changes in hash vectors should be weighted differently (to, say, downplay the impact of a copy-move object in a corner of the image).

4. Experiments

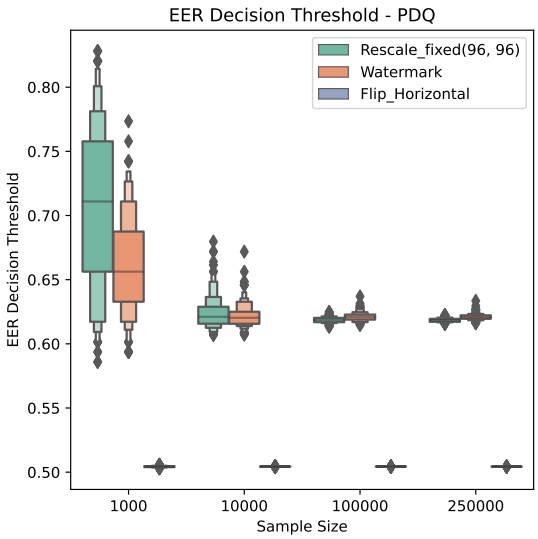
To demonstrate the utility of `PHASER`, we present two short experiments in this section, which explore open questions directly in the framework. The code used for these experiments serve as exemplars on the `PHASER` GitHub page.

4.1. How Many Images?: Convergence of Measures at Various Dataset Sizes

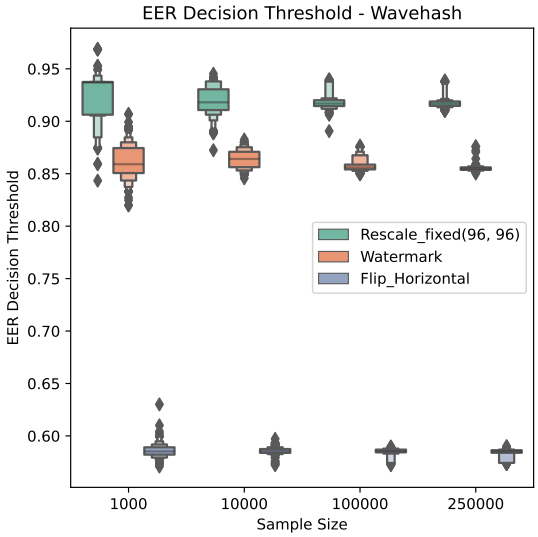
It was noted in Section 2.2 that the dataset of original images used to evaluate Perceptual Hashes varies considerably. When considering the use case of `PHASER`, it became necessary to establish how large the dataset must be to obtain reliable results. This information was not readily available in the literature, so an experiment was conducted to understand sampling variability.

The Flickr 1 Million dataset [37] was used due to its large scale, and all images were hashed using PDQ and Wavehash for the Flip_Horizontal (mirror on x-axis), Rescale (96×96px) and Watermark (40px min, target 10% image height, bottom right-corner) transforms. Prior work [28] has

shown that the inter-image distributions for PDQ and Wavehash vary considerably (narrow and wide distributions, respectively), making them complementary choices. For similar reasons, the transforms were chosen to represent an easy case (Rescale), a slightly more difficult case (Watermark), and a very difficult case (Flip_Horizontal), again based on prior work [28]. For the sake of simplicity, we only considered Hamming distance here, as it is the most widely used.



(a) EER Decision Threshold for PDQ over different dataset sample sizes.



(b) EER Decision Threshold for Wavehash over different dataset sample sizes.

Figure 4: Metric convergence for EER thresholds for three transforms. Note that the plots do not share the y-axis.

To simulate differing dataset scales, the DataFrame of all hashes was sampled for 1,000; 10,000; 100,000; and 250,000 images before calculating distance scores. Inter- and intra-distances and their corresponding evaluation metrics, were then calculated for each sample size, repeating the process 250 times for each. `PHASER` balances the number of inter- and intra-distance observations such that the number of observations for each inter- and intra-distance triplet is the same as the sample size. As this is a necessary step for some analysis approaches to be representative, we expect these results to hold for experiments conducted outside of `PHASER`, too.

To capture the variability of both inter- and intra-distance comparisons, we plot the EER Decision Threshold, i.e. the similarity value where both the FPR and FNR are equal, as it captures the relationship between inter- and intra-score distributions. The assumption is made that all original images in the dataset are unrelated for the sake of FPR/FNR calculation. The boxenplots for each algorithm are depicted in Figure 4.

The EER threshold for PDQ-Rescale (4a) shows considerable change from the smallest sample size, 1000, to the next order of magnitude, with the range shrinking from 0.59–0.83 to 0.61–0.68. This suggests that there are cases where there is a clear benefit to samples on the larger side of the literature, even for the least-difficult transform. The more difficult Watermark case has very similar behaviour, however, the difficult Flip transform does not change the EER threshold much at all, largely because PDQ is very poor at handling this transform, producing distances that correspond to unrelated images. Similar trends are found with Wavehash, though the absolute values vary considerably for the EER threshold, with a smaller maximum range.

Overall, the variance between iterations drops substantially as the dataset increases from 1,000 to 100,000 images. Table 1 depicts standard deviation values for the Rescale transform, as it seems to vary the most despite being a relatively trivial transformation. We note an order of magnitude difference between values at 1,000 and 100,000 samples, and there is still improvement to be had at 250,000. We would suggest that larger datasets may produce more consis-

Rescale(96, 96)	EER Threshold STDEV	
Sample Size	**PDQ**	**Wavehash**
1,000	0.0634	0.0212
10,000	0.0125	0.0134
100,000	0.0023	0.0076
250,000	0.0015	0.0047

Table 1: Standard deviations of EER-thresholds for the Rescale transform across dataset sizes. 250 iterations at each sample size.

tent results, though there could be diminishing returns beyond 100,000 on heterogeneous datasets. This point may occur much earlier for more constrained datasets, where images are relatively homogeneous.

4.2. Optimising for weaknesses: Assigning Hash Bit-Vector Weights

Perceptual hashing algorithms can be separated into heuristic approaches and machine learning-based approaches, which derive features or their respective weights directly from image datasets. However, aside from Steinebach's [38] weighting scheme for ForBild, data-driven optimisation of heuristic hashes does not appear to have been pursued. We explored this possibility by leveraging functionality in both `PHASER` and SciPy.

250,000 images were sampled from the Flickr 1 Million dataset, processing them with PDQ, pHash and Wavehash for a range of transforms: Border (30px), Crop (5% all sides), Flip (x-axis), Rescale(96x96), Rotate (anti-clockwise 5°), and Watermark. All three transforms are heuristic approaches and derive features in the frequency domain. These have been tested previously in prior work [10, 28, 36], with the x-axis mirroring proving to be the main point of difficulty.

Distance scores were then calculated for all triplets, with bit-analysis enabled when computing metrics. Each bit in the hash vector (varying in length for each algorithm) was analysed for its relative contribution to rejecting and accepting an image match correctly at the EER threshold. Median values are then used to generate a bit-weight vector using aggregated information across all transforms, ideally improving overall performance. Cosine distance was included in this analysis to determine whether potential benefits extend beyond Hamming distance.

Before Bit-weighting (All Transforms)					
Algorithm	**Distance Metric**	**AUC**		**EER**	
		mean	std	mean	std
PDQ	Hamming	0.918	0.197	0.090	0.199
	Cosine	0.918	0.197	0.090	0.199
pHash	Hamming	0.905	0.203	0.112	0.203
	Cosine	0.905	0.203	0.112	0.203
Wavehash	Hamming	0.817	0.335	0.188	0.310
	Cosine	0.817	0.335	0.188	0.310
After Bit-weighting (All Transforms)					
PDQ	Hamming	0.969	0.072	0.047	0.096
	Cosine	0.969	0.071	0.047	0.094
pHash	Hamming	0.931	0.129	0.090	0.143
	Cosine	0.932	0.128	0.088	0.140
Wavehash	Hamming	0.820	0.329	0.185	0.306
	Cosine	0.820	0.330	0.185	0.307

Table 2: A comparison of classification performance across aggregated transforms for various hashes. Measuring Area Under the ROC Curve (AUC), and Equal Error Rate (EER).

An aggregated performance summary across all transforms, before and after applying optimised weights, is provided in Table 2. Generally, all algorithms improve for distance and evaluation metrics, though the change is limited for Wavehash, while both PDQ and pHash see a larger benefit. Cosine distance values track Hamming closely for these hash types.

To determine if the aggregated benefit is derived from an overall improvement or, in difficult cases, the difference in AUC values between pre- and post-optimisation are plotted in Figure 5. The uplift in PDQ and pHash performance can be attributed to improving the difficult Flip case, while Wavehash seems to benefit slightly from the Border transform instead. On the other hand, pHash loses ground slightly in the Border case, though it still benefits significantly in the aggregated metrics.

To dive deeper, we plot the bit-analysis generated by `PHASER` for the pHash algorithm for the specific improvement case, Flip, and the aggregated rates across all transforms. Darker bars show when the bit more accurately matches the

ideal outcome, but a 50/50 chance is expected for inter-image comparisons.

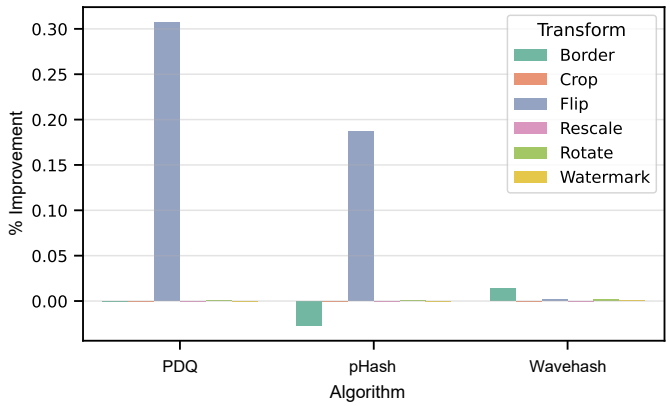
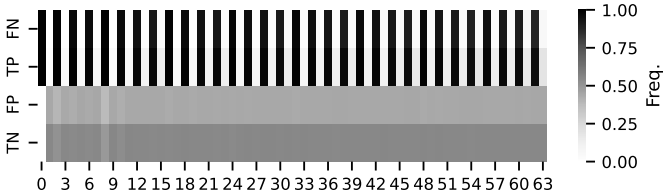
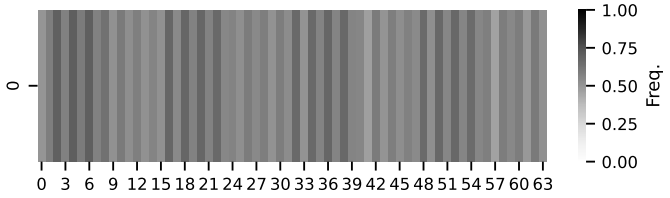


Figure 5: Differences in the AUC metric before and after applying bit-weights for each algorithm/transform for the normalised Hamming distance.



(a) Individual hash vector bit frequency rates in the pHash/Flip_Horizontal/Hamming triplet. Split by FN/TP/FP/TN.



(b) The aggregated (median) bit success rates for pHash across all transforms and success/failure classes. This is a visualisation of the calculated bit-weight vector.

Figure 6: Representations of the 'success rate' for each bit in the 64-bit pHash hash vector. A higher frequency means the bit matched more consistently with expectation, i.e., a difference for the inter-class and a match for the intra-class. The same patterns are present for PDQ, though it is easier to see with fewer bits.

Interestingly, a piano-like pattern is found in Figure 6a for the intra-image cases (TP/FN), where alternating bits switch between very high and very low degrees of accuracy. We suggest that this is an artefact of how these hashes derive the bits in their hash from the Discrete Cosine Transform (DCT) coefficient matrices. The upshot is that every second bit strongly matches the original and Flip transform. As such, re-balancing the weights to favour these, there is a large performance change for the Flip case. The part of the confusion matrix derived from inter-image comparisons is roughly as expected, though, at a rate of around 0.5.

The overall bit-weight applied to pHash for all comparisons is visualised in Figure 6b. the piano-like pattern to compensate for Flip is visible but is softened somewhat when median values from all transforms and success/failure classes are applied.

This result was quite unexpected, though it clearly demonstrates that algorithmic nuances may have unexpected interactions with particular transform classes. These insights can then provide wider knowledge for improving the algorithms or, in this case, for mitigating some of their weaknesses. It is also worth noting that there was almost no detrimental impact on performance for any given transform using the bit-weights, though a wider set of hashes and transform classes should be tested.

5. Conclusion and Future Work

This work described `PHASER`, a new open-sourced perceptual hashing evaluation framework, primarily targeting the digital forensics and content-detection use case. We then applied the framework to the Flick1m dataset to answer two open questions that need to be covered directly in the literature. Firstly, we observe that large datasets with >100k original images may provide more accurate results, and secondly, comparing perceptual hashes using learned weights may be useful. In particular, we discovered that the DCT-based transforms we tested have a peculiar behaviour on mirrored images, with half of the bits in the hash being strongly correlated with accurate classification and the other half strongly negatively correlated.

By open-sourcing and releasing `PHASER`, we hope to facilitate other researchers with the discovery of such anomalies and mitigations for them, either by forensics practitioners or scientists. Visual similarity matching is a difficult

task, but it is possible that existing elements of Content-Based Image Retrieval (CBIR) can be leveraged for this use case, for example, in multi-hash indices or via the development of more nuanced hash comparison techniques. Ultimately, there still appear to be multiple avenues to explore that may better scale detection in-line with demand.

References

[1] J. Lyle, "Computer forensic tool testing at nist," *URL http://www. cftt. nist. gov/documents/Amalfi-04. ppt, Accessed*, vol. 18, pp. 2003–30, 2007.

[2] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt, "Bringing science to digital forensics with standardized forensic corpora," *digital investigation*, vol. 6, pp. S2–S11, 2009.

[3] E. OliveiraJr, A. F. Zorzo, and C. V. Neu, "Towards a conceptual model for promoting digital forensics experiments," *Forensic Science International: Digital Investigation*, vol. 35, p. 301014, 2020.

[4] E. Casey, "The chequered past and risky future of digital forensics," *Australian journal of forensic sciences*, vol. 51, no. 6, pp. 649–664, 2019.

[5] G. Horsman, "Tool testing and reliability issues in the field of digital forensics," *Digital Investigation*, vol. 28, pp. 163–175, 2019.

[6] N. Beebe and J. Clark, "Dealing with terabyte data sets in digital investigations," in *Advances in Digital Forensics*, M. Pollitt and S. Shenoi, Eds. Boston, MA: Springer US, 2005, pp. 3–16.

[7] G. Michelet, F. Breitinger, and G. Horsman, "Automation for digital forensics: Towards a definition for the community," *Forensic Science International*, vol. 349, p. 111769, Aug. 2023.

[8] X. Du, C. Hargreaves, J. Sheppard, F. Anda, A. Sayakkara, N.-A. Le-Khac, and M. Scanlon, "Sok: Exploring the state of the art and the future potential of artificial intelligence in digital forensic investigation," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020, pp. 1–10.

[9] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital investigation*, vol. 3, pp. 91–97, 2006.

[10] F. Breitinger, H. Liu, C. Winter, H. Baier, A. Rybalchenko, and M. Steinebach, "Towards a process model for hash functions in digital forensics," in *International Conference on Digital Forensics and Cyber Crime*. Springer, 2013, pp. 170–186.

[11] E. Bursztein, E. Clarke, M. DeLaune, D. M. Elifff, N. Hsu, L. Olson, J. Shehan, M. Thakur, K. Thomas, and T. Bright, "Rethinking the detection of child sexual abuse imagery on the internet," in *The world wide web conference*, 2019, pp. 2601–2607.

[12] N. Krawetz. Photodna and limitations. Not known. [Online]. Available: https://www.hackerfactor.com/blog/index.php?archives/931-PhotoDNA-and-Limitations.html

[13] Facebook. PDQ. [Online]. Available: https://github.com/facebook/ThreatExchange/tree/main/pdq/python

[14] Apple, "Csam detection technical summary," https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf, 2022, [Online; accessed 2-Feb-2022].

[15] F. Breitinger, G. Stivaktakis, and H. Baier, "FRASH: A framework to test algorithms of similarity hashing," *Digital Investigation*, vol. 10, pp. S50–S58, Aug. 2013.

[16] T. Göbel, F. Uhlig, H. Baier, and F. Breitinger, "FRASHER – A framework for automated evaluation of similarity hashing," *Forensic Science International: Digital Investigation*, vol. 42, p. 301407, Jul. 2022.

[17] V. Tyagi, *Content-Based Image Retrieval*. Singapore: Springer Singapore, 2017.

[18] Q. Hao, L. Luo, S. T. Jan, and G. Wang, "It's Not What It Looks Like: Manipulating Perceptual Hashing based Applications," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. Virtual Event Republic of Korea: ACM, Nov. 2021, pp. 69–85.

[19] H. Müller, W. Müller, D. M. Squire, S. Marchand-Maillet, and T. Pun, "Performance evaluation in content-based image retrieval: overview and proposals," *Pattern recognition letters*, vol. 22, no. 5, pp. 593–601, 2001.

[20] C. Zauner, "Implementation and benchmarking of perceptual image hash functions," 2010.

[21] M. Steinebach, H. Liu, and Y. Yannikos, "ForBild: efficient robust image hashing," *Media Watermarking, Security, and Forensics 2012*, vol. 8303, p. 83030O, 2012.

[22] G. Horsman, "That tool is rubbish!...or is it?" *Science & Justice*, vol. 62, no. 5, pp. 515–519, Sep. 2022.

[23] A. Athalye, "Inverting PhotoDNA," 2021. [Online]. Available: https://anishathalye.com/inverting-photodna/

[24] J. Prokos, T. M. Jois, N. Fendley, R. Schuster, M. Green, E. Tromer, and Y. Cao, "Squint hard enough: Evaluating perceptual hashing with machine learning," Cryptology ePrint Archive, Paper 2021/1531, 2021. [Online]. Available: https://eprint.iacr.org/2021/1531

[25] S. Jain, A.-M. Creţu, and Y.-A. de Montjoye, "Adversarial Detection Avoidance Attacks: Evaluating the robustness of perceptual hashing-based client-side scanning," in *31st USENIX Security Symposium*

(USENIX Security 22), 2022, pp. 2317–2334.

[26] M. Steinebach, H. Liu, and Y. Yannikos, "Forbild: Efficient robust image hashing," in *Media Watermarking, Security, and Forensics 2012*, vol. 8303. SPIE, 2012, pp. 195–202.

[27] M. Hamadouche, K. Zebbiche, M. Guerroumi, H. Tebbi, and Y. Zafoune, "A comparative study of perceptual hashing algorithms: Application on fingerprint images," in *2nd International Conference on Computer Science's Complex Systems and their Application*, Algeria, May 2021, p. 12.

[28] S. McKeown and W. J. Buchanan, "Hamming distributions of popular perceptual hashing techniques," *Forensic Science International: Digital Investigation*, vol. 44, p. 301509, 2023.

[29] M. Ferenčak, P. Grd, and I. Tomičić, "The Impact of Image Processing on Perceptual Hash Values," in *2023 46th MIPRO ICT and Electronics Convention (MIPRO)*. Opatija, Croatia: IEEE, May 2023, pp. 1070–1075.

[30] M. Steinebach, H. Liu, and Y. Yannikos, "Efficient cropping-resistant robust image hashing," in *2014 Ninth International Conference on Availability, Reliability and Security*. IEEE, 2014, pp. 579–585.

[31] M. Gaillard and E. Egyed-Zsigmond, "Large scale reverse image search," *XXXVème Congrès INFORSID*, p. 127, 2017.

[32] S. McKeown, G. Russell, and P. Leimich, "Fast Forensic Triage Using Centralised Thumbnail Caches on Windows Operating Systems," *Journal of Digital Forensics, Security and Law*, vol. 14, no. 3, 2019.

[33] P. Samanta and S. Jain, "Analysis of Perceptual Hashing Algorithms in Image Manipulation Detection," *Procedia Computer Science*, vol. 185, pp. 203–212, 2021.

[34] M. Alkhowaiter, K. Almubarak, and C. Zou, "Evaluating perceptual hashing algorithms in detecting image manipulation over social media platforms," in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, 2022, pp. 149–156.

[35] B. Yang, F. Gu, and X. Niu, "Block mean value based image perceptual hashing," in *Intelligent Information Hiding and Multimedia Signal Processing, 2006. IIH-MSP'06. International Conference on*. IEEE, 2006, pp. 167–172.

[36] A. Drmic, M. Silic, G. Delac, K. Vladimir, and A. S. Kurdija, "Evaluating robustness of perceptual image hashing algorithms," in *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Opatija, Croatia: IEEE, May 2017, pp. 995–1000.

[37] "MIRFLICKR Download." [Online]. Available: http://press.liacs.nl/mirflickr/mirdownload.html

[38] M. Steinebach, "Robust hashing for efficient forensic analysis of image sets," in *International Conference on Digital Forensics and Cyber Crime*. Springer, 2011, pp. 180–187.

[39] J. Buchner. Image hash. [Online]. Available: https://github.com/JohannesBuchner/imagehash

[40] T.-J. Liu, W.-L. Tsao, and C.-L. Lee, "A high performance image-spam filtering system," in *2010 Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science*. IEEE, 2010, pp. 445–449.

[41] M. Steinebach, "An Analysis of PhotoDNA," in *Proceedings of the 18th International Conference on Availability, Reliability and Security*. Benevento Italy: ACM, Aug. 2023, pp. 1–8.