perception

Apr 28, 2020

Contents:

`perception` provides flexible, well-documented, and comprehensively tested tooling for perceptual hashing research, development, and production use. It provides a common wrapper around existing, popular perceptual hashes (such as those implemented by ImageHash) along with tools to compare their performance and use them for common tasks.

Perceptual hashes are used to create compact image "fingerprints" which are invariant to small alterations to the original image. Typically, the representations are compact enough that they are irreversible, which makes them useful for deduplication and detecting abusive content while preserving the privacy of content owners.

CHAPTER 1

Installation

You can install `perception` using pip. You must install OpenCV separately (e.g., with `pip install opencv-python`).

```
# Install from PyPi
pip install perception

# Install from GitHub
pip install git+https://github.com/thorn-oss/perception.git#egg=perception
```

To install with the necessary dependencies for benchmarking, use:

```
# Install from PyPi
pip install perception[benchmarking]

# Install from GitHub
pip install opencv-python git+https://github.com/thorn-oss/perception.git
↪#egg=perception[benchmarking]
```

CHAPTER 2

Getting Started

Please see the examples for code snippets for common use cases.

2.1 Examples

2.1.1 Media Deduplication

Perceptual hashes can be used to deduplicate sets of images. Below we provide two examples (one simple, one larger scale).

For most use cases, we recommend using PHash with `hash_size=16` and with 0.2 as the distance threshold as in the example below. You may wish to adjust this threshold up or down based on your tolerance for false negatives / positives.

In practice, deduplicating in memory on your machine by the methods below may be impractical. For larger-scale applications, you may wish to use tools like FAISS, Annoy, or databases with functionality for querying based on distance such as MemSQL.

For the supported hashers, below are our recommended thresholds with expected false positive rates of <1%.

hasher	threshold
ahash (hash_size=16)	0.008
blockmean	0.008
dhash (hash_size=16)	0.07
marrhildreth	0.1
pdq	0.2
phash (hash_size=16)	0.2
wavelet (hash_size=16)	0.02

Simple example

In this example, we download a ZIP file containing 18 images. One of the images is duplicated twice and another image is duplicated once.

```
import os
import glob
import zipfile
import urllib.request

import tabulate
import pandas as pd

from perception import tools, hashers

urllib.request.urlretrieve(
    "https://thorn-perception.s3.amazonaws.com/thorn-perceptual-deduplication-example.zip",
    "thorn-perceptual-deduplication-example.zip"
)

with zipfile.ZipFile('thorn-perceptual-deduplication-example.zip') as f:
    f.extractall('.')

filepaths = glob.glob('thorn-perceptual-deduplication-example/*.jpg')
duplicate_pairs = tools.deduplicate(files=filepaths, hashers=[(hashers.PHash(hash_size=16), 0.2)])
print(tabulate.tabulate(pd.DataFrame(duplicate_pairs), showindex=False, headers=['file1', 'file2'], tablefmt='rst'))

# Now we can do whatever we want with the duplicates. We could just delete
# the first entry in each pair or manually verify the pairs to ensure they
# are, in fact duplicates.
```

file1	file2
thorn-perceptual-deduplication-example/309b.jpg	thorn-perceptual-deduplication-example/309.jpg
thorn-perceptual-deduplication-example/309b.jpg	thorn-perceptual-deduplication-example/309a.jpg
thorn-perceptual-deduplication-example/309a.jpg	thorn-perceptual-deduplication-example/309.jpg
thorn-perceptual-deduplication-example/315a.jpg	thorn-perceptual-deduplication-example/315.jpg

Real-world example

In the example below, we use the Caltech 256 Categories dataset. Like most other public image datasets, it contains a handful of duplicates in some categories.

The code below will:

1. Download the dataset
2. Group all the filepaths by category (the dataset is provided in folders)
3. Within each group, find duplicates using PHash. We will compare not just the original images, but also the 8 isometric transformations for each image.

```
import os
import tarfile
```

(continues on next page)

(continued from previous page)

```
from glob import glob
import urllib.request

import tqdm

from perception import hashers, tools

urllib.request.urlretrieve(
    "http://www.vision.caltech.edu/Image_Datasets/Caltech256/256_ObjectCategories.tar
↪",
    "256_ObjectCategories.tar"
)
with tarfile.open('256_ObjectCategories.tar') as tfile:
    tfile.extractall()

files = glob('256_ObjectCategories/**/*.jpg')

# To reduce the number of pairwise comparisons,
# we can deduplicate within each image category
# (i.e., we don't need to compare images of
# butterflies with images of chess boards).
filepath_group = [
    (
        filepath,
        os.path.normpath(filepath).split(os.sep)[-2]
    ) for filepath in files
]
groups = list(set([group for _, group in filepath_group]))

# We consider any pair of images with a PHash distance of < 0.2 as
# as a duplicate.
comparison_hashers = [(hashers.PHash(hash_size=16), 0.2)]

duplicate_pairs = []

for current_group in groups:
    current_filepaths = [
        filepath for filepath, group in filepath_group if group == current_group
    ]
    current_duplicate_pairs = tools.deduplicate(
        files=current_filepaths,
        hashers=comparison_hashers,
        isometric=True,
        progress=tqdm.tqdm
    )
    duplicate_pairs.extend(current_duplicate_pairs)

# Now we can do whatever we want with the duplicates. We could just delete
# the first entry in each pair or manually verify the pairs to ensure they
# are, in fact duplicates.
```

Video deduplication

Video deduplication requires more thought depending on your tolerance for false positives and how important temporal relationships are. Below is one example approach for deduplicating a group of videos by taking frames from each video that are sufficiently different from each other (to avoid keeping too many) and then using them all to find pairs

of videos that have matching frames.

```
import urllib.request
import zipfile

import glob
import tqdm

import perception.hashers

# Download some example videos.
urllib.request.urlretrieve(
    "https://thorn-perception.s3.amazonaws.com/thorn-perceptual-video-deduplication-
↪example.zip",
    "thorn-perceptual-video-deduplication-example.zip"
)

with zipfile.ZipFile('thorn-perceptual-video-deduplication-example.zip') as f:
    f.extractall('.')

# By default, this will use TMK L1 with PHashU8.
hasher = perception.hashers.SimpleSceneDetection(max_scene_length=5)

# Set a threshold for matching frames within videos and across videos.
filepaths = glob.glob('thorn-perceptual-video-deduplication-example/*.m4v') + \
            glob.glob('thorn-perceptual-video-deduplication-example/*.gif')

# Returns a list of dicts with a "filepath" and "hash" key. "hash" contains a
# list of hashes.
hashes = hasher.compute_parallel(filepaths=filepaths, progress=tqdm.tqdm)


# Flatten the hashes into a list of (filepath, hash) tuples.
hashes_flattened = perception.tools.flatten([
    [(hash_group['filepath'], hash_string) for hash_string in hash_group['hash']]
    for hash_group in hashes
])

duplicates = perception.tools.deduplicate_hashes(
    hashes=hashes_flattened,
    threshold=50,
    hasher=hasher
)
```

2.1.2 Detecting Child Sexual Abuse Material

Using *perception* and a subscription to Thorn's Safer service, you can easily check for child sexual abuse material against a database of known bad content **without** having to send any images to a third party. You do this by sending compact, irreversible image hashes to get matches with a high degree of precision. We support matching using 16x16 PHash hashes and md5 hashes.

See usage example below. Please contact info@getsafer.io to discuss Thorn's Safer service and subscription options and visit getsafer.io to learn more.

```
from perception import tools
matcher = tools.SaferMatcher(
```

(continues on next page)

(continued from previous page)

```
    api_key='YOUR_API_KEY',
    url='MATCHING_SERVICE_URL'
)
matches = matcher.match(['myfile.jpg'])
```

In some cases, you may have a username/password instead of an API key, in which case you can pass those instead (see API documentation for details).

2.1.3 Benchmarking

This package provides a fair amount of infrastructure for benchmarking different hashers to evaluate their performance.

Image Hashing

The below example does the following:

- Download a benchmarking dataset (we provide a dataset with images that have compatible licensing for this example)
- Load the dataset. If you are using your own datasets, you may wish to call *deduplicate* on it to ensure no duplicates are included.
- Transform the dataset to generate synthetic images.
- Define a new custom hasher that we want to evaluate. It's not very good – but demonstrates how you can evaluate your own custom hash functions.
- Compute all the hashes.
- Report metrics for each image category / hasher / transformation combination.

```
import os
import glob
import zipfile
import urllib.request

import cv2
import imgaug
import tabulate # Optional: Only used for generating tables for the Sphinx␣
↪documentation
import numpy as np

from perception import benchmarking, hashers

urllib.request.urlretrieve(
    "https://thorn-perception.s3.amazonaws.com/thorn-perceptual-benchmark-v0.zip",
    "thorn-perceptual-benchmark-v0.zip"
)

with zipfile.ZipFile('thorn-perceptual-benchmark-v0.zip') as f:
    f.extractall('.')

# Load the dataset
dataset = benchmarking.BenchmarkImageDataset.from_tuples(files=[
    (filepath, filepath.split(os.path.sep)[-2]) for filepath in glob.glob(
        os.path.join('thorn-perceptual-benchmark-v0', '**', '*.jpg')
```

(continues on next page)

(continued from previous page)

```
    )
])

# Define the transforms we want to use for
# evaluation hash quality.
def watermark(image):
    fontScale = 5
    thickness = 5
    text = "TEXT"
    fontFace = cv2.FONT_HERSHEY_SIMPLEX
    targetWidth = 0.2*image.shape[1]
    (textWidth, textHeight), _ = cv2.getTextSize(
        text="TEST",
        fontFace=fontFace,
        fontScale=fontScale,
        thickness=thickness
    )
    fontScaleCorr = targetWidth / textWidth
    textHeight *= fontScaleCorr
    textWidth *= fontScaleCorr
    fontScale *= fontScaleCorr

    org = ( textHeight, image.shape[0] - textHeight )
    org = tuple(map(int, org))
    color = (0, 0, 0, 200)
    placeholder = cv2.putText(
        img=np.zeros(image.shape[:2] + (4, ), dtype='uint8'),
        text="TEST",
        org=org,
        color=color,
        fontFace=fontFace,
        fontScale=fontScale,
        thickness=thickness
    ).astype('float32')
    augmented = (
        (image.astype('float32')[..., :3]*(255 - placeholder[..., 3:]) + placeholder[.
↪.., :3]*placeholder[..., 3:])
    ) / 255
    return augmented.astype('uint8')

def vignette(image):
    height, width = image.shape[:2]
    a = cv2.getGaussianKernel(height, height/2)
    b = cv2.getGaussianKernel(width, width/2)
    c = (b.T*a)[..., np.newaxis]
    d = c/c.max()
    e = image*d
    return e.astype('uint8')

transforms={
    'watermark': watermark,
    'blur2': imgaug.augmenters.GaussianBlur(sigma=2.0),
    'vignette': vignette,
    'gamma2': imgaug.augmenters.GammaContrast(gamma=2),
    'jpeg95': imgaug.augmenters.JpegCompression(95),
    'pad0.2': imgaug.augmenters.Pad(percent=((0.2, 0.2), (0, 0), (0.2, 0.2), (0, 0)),␣
↪keep_size=False),
```

(continues on next page)

(continued from previous page)

```
    'crop0.05': imgaug.augmenters.Crop(percent=((0.05, 0.05), (0.05, 0.05), (0.05, 0.
↪05), (0.05, 0.05)), keep_size=False),
    'noise0.2': imgaug.augmenters.AdditiveGaussianNoise(scale=0.2*255),
    'rotate4': imgaug.augmenters.Affine(rotate=4),
    'noop': imgaug.augmenters.Resize({"longer-side": 256, "shorter-side": "keep-
↪aspect-ratio"}),
}

# Compute the transformed versions of the images.
# This takes a while but you can reload the
# generated dataset without recomputing it (see next line).
transformed = dataset.transform(
    transforms=transforms,
    storage_dir='transformed',
    errors="raise"
)
# We don't actually have to do this, but it shows
# how to reload the transformed dataset later.
transformed = benchmarking.BenchmarkImageTransforms.load(
    path_to_zip_or_directory='transformed', verify_md5=False
)

# Create a new hash that we want to evaluate.
# perception will handle most of the plumbing but
# we do have to specify a few things.
class ShrinkHash(hashers.Hasher):
    """This is a simple hash to demonstrate how you
    can create your own hasher and compare it to others.
    It just shrinks images to 8x8 pixels and then flattens
    the result.
    """

    # We have to let perception know
    # the shape and type of our hash.
    hash_length = 64
    dtype = 'uint8'

    # We need to specify how distance is
    # computed between hashes.
    distance_metric = 'euclidean'

    def _compute(self, image):
        gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        resized = cv2.resize(gray, dsize=(8, 8))
        return resized.flatten()

hashers_dict = {
    'ahash': hashers.AverageHash(hash_size=16),
    'dhash': hashers.DHash(hash_size=16),
    'pdq': hashers.PDQHash(),
    'phash': hashers.PHash(hash_size=16),
    'marrhildreth': hashers.MarrHildreth(),
    'wavelet': hashers.WaveletHash(hash_size=16),
    'blockmean': hashers.BlockMean(),
    'shrinkhash': ShrinkHash()
}
```

(continues on next page)

(continued from previous page)

```
# Compute the hashes
hashes = transformed.compute_hashes(hashers=hashers_dict)

# Get performance metrics (i.e., recall) for each hash function based on
# a false positive rate tolerance threshold. Here we use 0.01%
fpr_threshold = 1e-4

# The metrics are just pandas dataframes. We use tabulate here to obtain the tables
# formatted for the documentation.
metrics = hashes.compute_threshold_recall(fpr_threshold=fpr_threshold).reset_index()
print(tabulate.tabulate(metrics, showindex=False, headers=metrics.columns, tablefmt=
↪'rst'))

metrics_by_transform = hashes.compute_threshold_recall(grouping=['transform_name'],␣
↪fpr_threshold=fpr_threshold).reset_index()
print(tabulate.tabulate(metrics_by_transform, showindex=False, headers=metrics_by_
↪transform.columns, tablefmt='rst'))

metrics_simple = hashes.compute_threshold_recall(grouping=[], fpr_threshold=fpr_
↪threshold).reset_index()
print(tabulate.tabulate(metrics_simple, showindex=False, headers=metrics_simple.
↪columns, tablefmt='rst'))
```

category	transform_name	hasher_name	threshold	recall	fpr	n_exemplars
paintings	blur2	ahash	0.0117188	66.062	0	2204
paintings	blur2	blockmean	0.0134298	87.432	0	2204
paintings	blur2	dhash	0.132812	100	0	2204
paintings	blur2	marrhildreth	0.126736	100	0	2204
paintings	blur2	pdq	0.117188	100	0	2204
paintings	blur2	phash	0.09375	100	0	2204
paintings	blur2	shrinkhash	61.441	43.829	0	2204
paintings	blur2	wavelet	0.015625	65.926	0	2204
paintings	crop0.05	ahash	0.0078125	0.227	0	2204
paintings	crop0.05	blockmean	0.0144628	0.408	0	2204
paintings	crop0.05	dhash	0.222656	11.298	0	2204
paintings	crop0.05	marrhildreth	0.215278	3.857	0	2204
paintings	crop0.05	pdq	0.265625	11.298	0	2204
paintings	crop0.05	phash	0.234375	8.757	0	2204
paintings	crop0.05	shrinkhash	95.5667	2.314	0	2204
paintings	crop0.05	wavelet	0.015625	0.318	0	2204
paintings	gamma2	ahash	0.0078125	2.586	0	2204
paintings	gamma2	blockmean	0.00826446	2.269	0	2204
paintings	gamma2	dhash	0.175781	98.82	0	2204
paintings	gamma2	marrhildreth	0.163194	99.501	0	2204
paintings	gamma2	pdq	0.164062	100	0	2204
paintings	gamma2	phash	0.164062	100	0	2204
paintings	gamma2	shrinkhash	180.69	0.045	0	2204
paintings	gamma2	wavelet	0.015625	18.603	0	2204
paintings	jpeg95	ahash	0.0117188	29.9	0	2204
paintings	jpeg95	blockmean	0.0134298	38.612	0	2204
paintings	jpeg95	dhash	0.191406	92.604	0	2204
paintings	jpeg95	marrhildreth	0.166667	85.844	0	2204

Continued on next page

Table 1 – continued from previous page

category	transform_name	hasher_name	threshold	recall	fpr	n_exemplars
paintings	jpeg95	pdq	0.25	100	0	2204
paintings	jpeg95	phash	0.25	100	0	2204
paintings	jpeg95	shrinkhash	66.7008	46.597	0	2204
paintings	jpeg95	wavelet	0.015625	19.419	0	2204
paintings	noise0.2	ahash	0.0078125	6.352	0	2204
paintings	noise0.2	blockmean	0.0154959	21.779	0	2204
paintings	noise0.2	dhash	0.238281	90.699	0	2204
paintings	noise0.2	marrhildreth	0.166667	72.096	0	2204
paintings	noise0.2	pdq	0.28125	99.501	0	2204
paintings	noise0.2	phash	0.273438	99.909	0	2204
paintings	noise0.2	shrinkhash	154.729	0.635	0	2204
paintings	noise0.2	wavelet	0.0078125	1.407	0	2204
paintings	noop	ahash	0	100	0	2204
paintings	noop	blockmean	0	100	0	2204
paintings	noop	dhash	0	100	0	2204
paintings	noop	marrhildreth	0	100	0	2204
paintings	noop	pdq	0	100	0	2204
paintings	noop	phash	0	100	0	2204
paintings	noop	shrinkhash	0	100	0	2204
paintings	noop	wavelet	0	100	0	2204
paintings	pad0.2	ahash	0.0820312	0.045	0	2204
paintings	pad0.2	blockmean	0.0950413	0.045	0	2204
paintings	pad0.2	dhash	0.214844	1.27	0	2204
paintings	pad0.2	marrhildreth	0.220486	0.045	0	2204
paintings	pad0.2	pdq	0.296875	2.586	0	2204
paintings	pad0.2	phash	0.28125	3.448	0	2204
paintings	pad0.2	shrinkhash	153.981	0.227	0	2204
paintings	pad0.2	wavelet	0.109375	0	0	2204
paintings	rotate4	ahash	0.0429688	4.083	0	2204
paintings	rotate4	blockmean	0.0392562	3.448	0	2204
paintings	rotate4	dhash	0.210938	40.245	0	2204
paintings	rotate4	marrhildreth	0.229167	64.201	0	2204
paintings	rotate4	pdq	0.28125	61.388	0	2204
paintings	rotate4	phash	0.265625	66.924	0	2204
paintings	rotate4	shrinkhash	69.4622	2.858	0	2204
paintings	rotate4	wavelet	0.0390625	0.635	0	2204
paintings	vignette	ahash	0.046875	7.623	0	2204
paintings	vignette	blockmean	0.0485537	8.53	0	2204
paintings	vignette	dhash	0.125	34.256	0	2204
paintings	vignette	marrhildreth	0.177083	77.813	0	2204
paintings	vignette	pdq	0.132812	100	0	2204
paintings	vignette	phash	0.132812	100	0	2204
paintings	vignette	shrinkhash	103.015	3.312	0	2204
paintings	vignette	wavelet	0.0546875	5.172	0	2204
paintings	watermark	ahash	0.0078125	31.307	0	2204
paintings	watermark	blockmean	0.0134298	47.55	0	2204
paintings	watermark	dhash	0.0664062	100	0	2204
paintings	watermark	marrhildreth	0.0711806	100	0	2204
paintings	watermark	pdq	0.28125	99.138	0	2204

Continued on next page

Table 1 – continued from previous page

category	transform_name	hasher_name	threshold	recall	fpr	n_exemplars
paintings	watermark	phash	0.289062	99.682	0	2204
paintings	watermark	shrinkhash	104.723	75.635	0	2204
paintings	watermark	wavelet	0.015625	51.18	0	2204
photographs	blur2	ahash	0.0195312	80.788	0	1650
photographs	blur2	blockmean	0.0330579	97.818	0	1650
photographs	blur2	dhash	0.0898438	96.303	0	1650
photographs	blur2	marrhildreth	0.102431	96.97	0	1650
photographs	blur2	pdq	0.304688	99.939	0	1650
photographs	blur2	phash	0.179688	100	0	1650
photographs	blur2	shrinkhash	116.09	42.303	0	1650
photographs	blur2	wavelet	0.0234375	78.303	0	1650
photographs	crop0.05	ahash	0.0117188	0.242	0	1650
photographs	crop0.05	blockmean	0.0278926	0.848	0	1650
photographs	crop0.05	dhash	0.101562	1.333	0	1650
photographs	crop0.05	marrhildreth	0.175347	3.152	0	1650
photographs	crop0.05	pdq	0.320312	38.485	0	1650
photographs	crop0.05	phash	0.335938	73.394	0	1650
photographs	crop0.05	shrinkhash	128.222	1.212	0	1650
photographs	crop0.05	wavelet	0.0234375	0.424	0	1650
photographs	gamma2	ahash	0.0195312	10.606	0	1650
photographs	gamma2	blockmean	0.0278926	18.242	0	1650
photographs	gamma2	dhash	0.105469	91.636	0	1650
photographs	gamma2	marrhildreth	0.121528	92.303	0	1650
photographs	gamma2	pdq	0.195312	100	0	1650
photographs	gamma2	phash	0.234375	100	0	1650
photographs	gamma2	shrinkhash	121.569	0.545	0	1650
photographs	gamma2	wavelet	0.0234375	19.152	0	1650
photographs	jpeg95	ahash	0.0117188	33.576	0	1650
photographs	jpeg95	blockmean	0.0299587	84.424	0	1650
photographs	jpeg95	dhash	0.117188	77.273	0	1650
photographs	jpeg95	marrhildreth	0.109375	73.333	0	1650
photographs	jpeg95	pdq	0.4375	99.939	0	1650
photographs	jpeg95	phash	0.335938	99.879	0	1650
photographs	jpeg95	shrinkhash	124.78	83.758	0	1650
photographs	jpeg95	wavelet	0.0234375	44.727	0	1650
photographs	noise0.2	ahash	0.0195312	34.909	0	1650
photographs	noise0.2	blockmean	0.036157	72.121	0	1650
photographs	noise0.2	dhash	0.167969	69.03	0	1650
photographs	noise0.2	marrhildreth	0.119792	56.182	0	1650
photographs	noise0.2	pdq	0.34375	99.758	0	1650
photographs	noise0.2	phash	0.320312	99.818	0	1650
photographs	noise0.2	shrinkhash	190.137	24	0	1650
photographs	noise0.2	wavelet	0.0234375	23.03	0	1650
photographs	noop	ahash	0	100	0	1650
photographs	noop	blockmean	0	100	0	1650
photographs	noop	dhash	0	100	0	1650
photographs	noop	marrhildreth	0	100	0	1650
photographs	noop	pdq	0	100	0	1650
photographs	noop	phash	0	100	0	1650

Continued on next page

Table 1 – continued from previous page

category	transform_name	hasher_name	threshold	recall	fpr	n_exemplars
photographs	noop	shrinkhash	0	100	0	1650
photographs	noop	wavelet	0	100	0	1650
photographs	pad0.2	ahash	0.046875	0.121	0	1650
photographs	pad0.2	blockmean	0.0588843	0.061	0	1650
photographs	pad0.2	dhash	0.109375	0.667	0	1650
photographs	pad0.2	marrhildreth	0.190972	0.182	0	1650
photographs	pad0.2	pdq	0.289062	1.515	0	1650
photographs	pad0.2	phash	0.296875	4.606	0	1650
photographs	pad0.2	shrinkhash	164.593	0.121	0	1650
photographs	pad0.2	wavelet	0.0820312	0	0	1650
photographs	rotate4	ahash	0.03125	2.545	0	1650
photographs	rotate4	blockmean	0.0382231	4.242	0	1650
photographs	rotate4	dhash	0.0976562	3.333	0	1650
photographs	rotate4	marrhildreth	0.159722	7.394	0	1650
photographs	rotate4	pdq	0.3125	78.121	0	1650
photographs	rotate4	phash	0.320312	92.182	0	1650
photographs	rotate4	shrinkhash	132.944	4.788	0	1650
photographs	rotate4	wavelet	0.015625	0.182	0	1650
photographs	vignette	ahash	0.03125	9.152	0	1650
photographs	vignette	blockmean	0.0330579	10.242	0	1650
photographs	vignette	dhash	0.0742188	24.606	0	1650
photographs	vignette	marrhildreth	0.0954861	38.606	0	1650
photographs	vignette	pdq	0.117188	100	0	1650
photographs	vignette	phash	0.125	100	0	1650
photographs	vignette	shrinkhash	133.364	10.727	0	1650
photographs	vignette	wavelet	0.0234375	4.424	0	1650
photographs	watermark	ahash	0.0195312	48	0	1650
photographs	watermark	blockmean	0.0258264	59.697	0	1650
photographs	watermark	dhash	0.078125	100	0	1650
photographs	watermark	marrhildreth	0.114583	98.242	0	1650
photographs	watermark	pdq	0.351562	99.879	0	1650
photographs	watermark	phash	0.320312	99.758	0	1650
photographs	watermark	shrinkhash	142.317	78.242	0	1650
photographs	watermark	wavelet	0.0234375	51.515	0	1650

transform_name	hasher_name	threshold	recall	fpr	n_exemplars
blur2	ahash	0.0117188	62.247	0	3854
blur2	blockmean	0.0134298	82.045	0	3854
blur2	dhash	0.0898438	98.054	0	3854
blur2	marrhildreth	0.102431	98.651	0	3854
blur2	pdq	0.304688	99.974	0	3854
blur2	phash	0.179688	100	0	3854
blur2	shrinkhash	61.441	28.23	0	3854
blur2	wavelet	0.015625	59.964	0	3854
crop0.05	ahash	0.0078125	0.208	0	3854
crop0.05	blockmean	0.0144628	0.337	0	3854
crop0.05	dhash	0.101562	0.597	0	3854
crop0.05	marrhildreth	0.175347	1.635	0	3854

Continued on next page

Table 2 – continued from previous page

transform_name	hasher_name	threshold	recall	fpr	n_exemplars
crop0.05	pdq	0.265625	11.598	0	3854
crop0.05	phash	0.234375	9.185	0	3854
crop0.05	shrinkhash	95.5667	1.427	0	3854
crop0.05	wavelet	0.015625	0.259	0	3854
gamma2	ahash	0.0078125	2.647	0	3854
gamma2	blockmean	0.00826446	2.335	0	3854
gamma2	dhash	0.105469	91.048	0	3854
gamma2	marrhildreth	0.121528	95.381	0	3854
gamma2	pdq	0.195312	100	0	3854
gamma2	phash	0.234375	100	0	3854
gamma2	shrinkhash	112.911	0.182	0	3854
gamma2	wavelet	0.015625	15.153	0	3854
jpeg95	ahash	0.0117188	31.474	0	3854
jpeg95	blockmean	0.0134298	39.673	0	3854
jpeg95	dhash	0.117188	64.037	0	3854
jpeg95	marrhildreth	0.109375	66.762	0	3854
jpeg95	pdq	0.273438	99.87	0	3854
jpeg95	phash	0.335938	99.948	0	3854
jpeg95	shrinkhash	66.7008	33.083	0	3854
jpeg95	wavelet	0.015625	21.069	0	3854
noise0.2	ahash	0.0078125	7.421	0	3854
noise0.2	blockmean	0.0154959	23.638	0	3854
noise0.2	dhash	0.167969	63.83	0	3854
noise0.2	marrhildreth	0.119792	46.341	0	3854
noise0.2	pdq	0.28125	99.559	0	3854
noise0.2	phash	0.273438	99.87	0	3854
noise0.2	shrinkhash	154.729	0.934	0	3854
noise0.2	wavelet	0.0078125	1.635	0	3854
noop	ahash	0	100	0	3854
noop	blockmean	0	100	0	3854
noop	dhash	0	100	0	3854
noop	marrhildreth	0	100	0	3854
noop	pdq	0	100	0	3854
noop	phash	0	100	0	3854
noop	shrinkhash	0	100	0	3854
noop	wavelet	0	100	0	3854
pad0.2	ahash	0.046875	0.052	0	3854
pad0.2	blockmean	0.0588843	0.026	0	3854
pad0.2	dhash	0.109375	0.285	0	3854
pad0.2	marrhildreth	0.190972	0.104	0	3854
pad0.2	pdq	0.289062	1.738	0	3854
pad0.2	phash	0.28125	3.269	0	3854
pad0.2	shrinkhash	136.11	0.078	0	3854
pad0.2	wavelet	0.0820312	0	0	3854
rotate4	ahash	0.03125	1.946	0	3854
rotate4	blockmean	0.0382231	3.503	0	3854
rotate4	dhash	0.0976562	1.583	0	3854
rotate4	marrhildreth	0.159722	6.046	0	3854
rotate4	pdq	0.28125	60.042	0	3854

Continued on next page

Table 2 – continued from previous page

transform_name	hasher_name	threshold	recall	fpr	n_exemplars
rotate4	phash	0.265625	65.646	0	3854
rotate4	shrinkhash	69.4622	1.92	0	3854
rotate4	wavelet	0.015625	0.078	0	3854
vignette	ahash	0.03125	5.475	0	3854
vignette	blockmean	0.0330579	6.461	0	3854
vignette	dhash	0.0742188	14.011	0	3854
vignette	marrhildreth	0.0954861	30.436	0	3854
vignette	pdq	0.132812	100	0	3854
vignette	phash	0.132812	100	0	3854
vignette	shrinkhash	103.015	4.515	0	3854
vignette	wavelet	0.0234375	2.024	0	3854
watermark	ahash	0.0078125	28.464	0	3854
watermark	blockmean	0.0134298	43.15	0	3854
watermark	dhash	0.078125	100	0	3854
watermark	marrhildreth	0.114583	99.248	0	3854
watermark	pdq	0.28125	99.325	0	3854
watermark	phash	0.289062	99.481	0	3854
watermark	shrinkhash	104.666	70.239	0	3854
watermark	wavelet	0.015625	46.653	0	3854

hasher_name	threshold	recall	fpr	n_exemplars
ahash	0.0078125	20.005	0	38540
blockmean	0.00826446	22.003	0	38540
dhash	0.0898438	46.798	6.07681e-05	38540
marrhildreth	0.102431	52.377	9.97855e-05	38540
pdq	0.265625	75.846	6.93433e-05	38540
phash	0.273438	80.106	6.56685e-05	38540
shrinkhash	60.1166	19.538	0	38540
wavelet	0.0078125	16.168	0	38540

Video Hashing

The below example does the following:

- Download a benchmarking dataset. Here we use the Charades dataset which contain over 9,000 videos.
- Load the dataset.
- Transform the dataset to generate synthetically altered videos. Our hashers are responsible for matching the altered videos with the originals.
- Define some hashers we want to evaluate.
- Compute all the hashes.
- Report metrics for each video category / hasher / transformation combination to see how well our hashers can match the altered videos to the original ("no-op" videos).

```
import os
import zipfile
import urllib.request
```

(continues on next page)

(continued from previous page)

```
import pandas as pd

import perception.benchmarking
import perception.hashers

if not os.path.isdir('Charades_v1_480'):
    # Download the dataset since it appears we do not have it. Note that
    # these are large files (> 13GB).
    urllib.request.urlretrieve(
        url='http://ai2-website.s3.amazonaws.com/data/Charades_v1_480.zip',
        filename='Charades_v1_480.zip'
    )
    with zipfile.ZipFile('Charades_v1_480.zip') as zfile:
        zfile.extractall('.')
    urllib.request.urlretrieve(
        url='http://ai2-website.s3.amazonaws.com/data/Charades.zip',
        filename='Charades.zip'
    )
    with zipfile.ZipFile('Charades.zip') as zfile:
        zfile.extractall('.')


# These are files that we've identified as having identical subsequences, typically
# when a person is out of frame and the backgrounds are the same.
duplicates = [
    ('0HVVN.mp4', 'UZRQD.mp4'), ('ZIOET.mp4', 'YGXX6.mp4'), ('82XPD.mp4', 'E7QDZ.mp4
↪'),
    ('FQDS1.mp4', 'AIOTI.mp4'), ('PBV4T.mp4', 'XXYWL.mp4'), ('M0P0H.mp4', 'STY6W.mp4
↪'),
    ('3Q92U.mp4', 'GHPO3.mp4'), ('NFIQM.mp4', 'I2DHG.mp4'), ('PIRMO.mp4', '0GFE8.mp4
↪'),
    ('LRPBA.mp4', '9VK0J.mp4'), ('UI0QG.mp4', 'FHXKQ.mp4'), ('Y05U8.mp4', '4RVZB.mp4
↪'),
    ('J6TVB.mp4', '2ZBL5.mp4'), ('A8T8V.mp4', 'IGOQK.mp4'), ('H8QM1.mp4', 'QYMWC.mp4
↪'),
    ('O45BC.mp4', 'ZS7X6.mp4'), ('NOP6W.mp4', 'F7KFE.mp4'), ('4MPPQ.mp4', 'A3M94.mp4
↪'),
    ('L8FFR.mp4', 'M8MP0.mp4'), ('EHYXP.mp4', 'O8PO3.mp4'), ('MGBLJ.mp4', 'RIEG6.mp4
↪'),
    ('53FPM.mp4', 'BLFEV.mp4'), ('UIIF3.mp4', 'TKEKQ.mp4'), ('GVX7E.mp4', '7GPSY.mp4
↪'),
    ('T7HZB.mp4', '6KGZA.mp4'), ('65M4K.mp4', 'UDGP2.mp4'), ('6SS4H.mp4', 'CK6OL.mp4
↪'),
    ('OVHFT.mp4', 'GG1X2.mp4'), ('VEHER.mp4', 'XBPEJ.mp4'), ('WN38A.mp4', '2QI8F.mp4
↪'),
    ('UMXKN.mp4', 'EOKJ0.mp4'), ('OSIKP.mp4', 'WT2C0.mp4'), ('H5V2Y.mp4', 'ZXN6A.mp4
↪'),
    ('XS6PF.mp4', '1WJ6O.mp4'), ('S2XJW.mp4', 'YH0BX.mp4'), ('UO607.mp4', 'Z5JZD.mp4
↪'),
    ('XN64E.mp4', 'CSRZM.mp4'), ('YXI7M.mp4', 'IKQLJ.mp4'), ('1B9C8.mp4', '004QE.mp4
↪'),
    ('V1SQH.mp4', '48WOM.mp4'), ('107YZ.mp4', 'I049A.mp4'), ('3S6WL.mp4', 'SC5YW.mp4
↪'),
    ('OY50Q.mp4', '5T607.mp4'), ('XKH7W.mp4', '028CE.mp4'), ('X8XQE.mp4', 'J0VXY.mp4
↪'),
    ('STB0G.mp4', 'J0VXY.mp4'), ('UNXLF.mp4', 'J0VXY.mp4'), ('56PK0.mp4', 'M1TZR.mp4
↪'),
```

(continues on next page)

(continued from previous page)

```
    ('FVITB.mp4', 'R0M34.mp4'), ('BPZE3.mp4', 'R0M34.mp4'), ('VS7DA.mp4', '1X0M3.mp4
↪'),
    ('I7MEA.mp4', 'YMM1Z.mp4'), ('9N76L.mp4', '0LDP7.mp4'), ('AXS82.mp4', 'W8WRK.mp4
↪'),
    ('8TSU4.mp4', 'MXATD.mp4'), ('80FWF.mp4', '18HFG.mp4'), ('RO3A2.mp4', 'V4HY4.mp4
↪'),
    ('HU409.mp4', 'BDWIX.mp4'), ('3YY88.mp4', 'EHHRS.mp4'), ('65RS3.mp4', 'SLIH4.mp4
↪'),
    ('LR0L8.mp4', 'Y665P.mp4')
]

blacklist = [fp1 for fp1, fp2 in duplicates]
df = pd.concat([pd.read_csv('Charades/Charades_v1_test.csv'), pd.read_csv('Charades/
↪Charades_v1_train.csv')])
df = df[~(df['id'] + '.mp4').isin(blacklist)]
df['filepath'] = df['id'].apply(lambda video_id: os.path.join('Charades_v1_480',␣
↪video_id + '.mp4'))
assert df['filepath'].apply(os.path.isfile).all(), 'Some video files are missing.'
dataset = perception.benchmarking.BenchmarkVideoDataset.from_tuples(
    files=df[['filepath', 'scene']].itertuples(index=False)
)

if not os.path.isdir('benchmarking_videos'):
    # We haven't computed the transforms yet, so we do that
    # now. Below, we create the following files for each of
    # the videos in our dataset. Note that the only required
    # transform is `noop` (see documentation for
    # perception.bencharmking.BenchmarkVideoDataset.transform).
    #
    # noop: This is the base video we'll actually use in benchmarking, rather
    #       than using the raw video. It is the same as the raw video but downsampled
    #       to a size that is reasonable for hashing (240p). This is because all
    #       of our hashers downsample to a size smaller than this anyway, so there
    #       is no benefit to a higher resolution. Also, we limit the length to the
    #       first five minutes of the video, which speeds everything up significantly.
    # shrink: Shrink the noop video down to 70% of its original size.
    # clip0.2: Clip the first 20% and last 20% of the noop video off.
    # slideshow: Create a slideshow version of the video that grabs frames␣
↪periodically
    #            from the original.
    # black_frames: Add black frames before and after the start of the video.
    # gif: Create a GIF from the video (similar to slideshow but with re-encoding)
    # black_padding: Add black bars to the top and bottom of the video.
    pad_width = 240
    pad_height = 320
    transforms = {
        'noop': perception.benchmarking.video_transforms.get_simple_transform(
            width='ceil(min(240/max(iw, ih), 1)*iw/2)*2',
            height='ceil(min(240/max(iw, ih), 1)*ih/2)*2',
            codec='h264',
            output_ext='.m4v',
            sar='1/1',
            clip_s=(None, 60*5)
        ),
        'shrink': perception.benchmarking.video_transforms.get_simple_transform(
            width='ceil(0.7*iw/2)*2',
            height='ceil(0.7*ih/2)*2'
```

(continues on next page)

(continued from previous page)

```
        ),
        'clip0.2': perception.benchmarking.video_transforms.get_simple_transform(clip_
↪pct=(0.2, 0.8)),
        'slideshow': perception.benchmarking.video_transforms.get_slideshow_transform(
            frame_input_rate=1/2.5, frame_output_rate=0.5, max_frames=10, offset=1.3),
        'black_frames': perception.benchmarking.video_transforms.get_black_frame_
↪padding_transform(0.5, 0.05),
        'gif': perception.benchmarking.video_transforms.get_simple_transform(
            output_ext='.gif', codec='gif', clip_s=(1.2, 10.2), fps=1/2.5
        ),
        'black_padding': perception.benchmarking.video_transforms.get_simple_
↪transform(
            width=f'(iw*sar)*min({pad_width}/(iw*sar),{pad_height}/ih)', height=f
↪'ih*min({pad_width}/(iw*sar),{pad_height}/ih)',
            pad=f'{pad_width}:{pad_height}:({pad_width}-iw*min({pad_width}/iw,{pad_
↪height}/ih))/2:({pad_height}-ih*min({pad_width}/iw,{pad_height}/ih))/2'
        )
    }

    # Save the transforms for later.
    transformed = dataset.transform(transforms=transforms, storage_dir='benchmarking_
↪videos')

transformed = perception.benchmarking.BenchmarkVideoTransforms.load('benchmarking_
↪videos', verify_md5=False)

phashu8 = perception.hashers.PHashU8(exclude_first_term=False, freq_shift=1, hash_
↪size=12)
hashers = {
    'phashu8_framewise': perception.hashers.FramewiseHasher(
        frames_per_second=1, frame_hasher=phashu8, interframe_threshold=50, quality_
↪threshold=90),
    'phashu8_tmkl1': perception.hashers.SimpleSceneDetection(
        base_hasher=perception.hashers.TMKL1(
            frames_per_second=5, frame_hasher=phashu8,
            distance_metric='euclidean', dtype='uint8',
            norm=None, quality_threshold=90),
        max_scene_length=1,
        interscene_threshold=50
    )
}
if not os.path.isfile('hashes.csv'):
    # We haven't computed the hashes, so we do that now.
    hashes = transformed.compute_hashes(hashers=hashers, max_workers=0)
    # Save the hashes for later. It took a long time after all!
    hashes.save('hashes.csv')

hashes = perception.benchmarking.BenchmarkHashes.load('hashes.csv')

hashes.compute_threshold_recall(fpr_threshold=0.001, grouping=['transform_name'])
```

transform_name	hasher_name	threshold	recall	fpr	n_exemplars
black_frames	phashu8_framewise	51.0979	88.163	0.000933489	277865
black_frames	phashu8_tmkl1	55.7584	99.918	0.000821862	403415
black_padding	phashu8_framewise	74.6391	7.689	0	276585
black_padding	phashu8_tmkl1	53.8702	99.887	0.000924784	411664
clip0.2	phashu8_framewise	54.8635	90.772	0.000904977	223591
clip0.2	phashu8_tmkl1	59.1693	99.753	0.000926021	323870
gif	phashu8_framewise	55.4437	68.314	0.000913103	82038
gif	phashu8_tmkl1	63.773	82.926	0.000993172	32140
noop	phashu8_framewise	0	100	0	281976
noop	phashu8_tmkl1	0	100	0	408673
shrink	phashu8_framewise	24.7184	100	0	280617
shrink	phashu8_tmkl1	52.8678	99.866	0.000926307	399357
slideshow	phashu8_framewise	56.9825	99.712	0.000926689	164361
slideshow	phashu8_tmkl1	63.4271	95.131	0.000988576	71668

2.2 API

2.2.1 Hashers

All hashers from the `Hasher` class.

class `perception.hashers.hasher.`**Hasher**

All hashers implement a common set of methods from the Hasher base class.

allow_parallel = True

Indicates whether the hashes can be computed in parallel

compute_distance(*hash1*, *hash2*, *hash_format='base64'*)

Compute the distance between two hashes.

Parameters

- **hash1** (`Union`[`ndarray`, `str`]) – The first hash or vector
- **hash2** (`Union`[`ndarray`, `str`]) – The second hash or vector
- **hash_format** – If either or both of the hashes are hash strings, what format the string is encoded in.

compute_parallel(*filepaths*, *progress=None*, *progress_desc=None*, *max_workers=5*, *isometric=False*)

Compute hashes in a parallelized fashion.

Parameters

- **filepaths** – A list of paths to images or videos (depending on the hasher).
- **progress** – A tqdm-like wrapper for reporting progress. If None, progress is not reported.
- **progress_desc** – The title of the progress bar.
- **max_workers** – The maximum number of workers
- **isometric** – Whether to compute all eight isometric transforms for each image.

distance_metric = None
The metric to use when computing distance between two hashes. All hashers must supply this parameter.

dtype = None
The numpy type to use when converting from string to array form. All hashers must supply this parameter.

hash_length = None
Indicates the length of the hash vector

returns_multiple = False
Whether or not this hash returns multiple values

string_to_vector(*hash_string*, *hash_format='base64'*)
Convert hash string to vector.

Parameters

- **hash_string** (str) – The input hash string
- **hash_format** (str) – One of 'base64' or 'hex'

vector_to_string(*vector*, *hash_format='base64'*)
Convert vector to hash string.

Parameters

- **vector** (ndarray) – Input vector
- **hash_format** (str) – One of 'base64' or 'hex'

Images

All image hashers inherit from the ImageHasher class.

class perception.hashers.hasher.**ImageHasher**

compute(*image*, *hash_format='base64'*)
Compute a hash from an image.

Parameters

- **image** (Union[str, ndarray, Image, BytesIO]) – An image represented as a filepath, a PIL image object, or as an np.ndarray object. If it is an np.ndarray object, it must be in RGB color order (note the OpenCV default is BGR).
- **hash_format** – One 'base64', 'hex', or 'vector'

Return type Union[str, ndarray]

compute_isometric_from_hash(*hash_string_or_vector*, *hash_format='base64'*)
For supported hashes, obtain the hashes for the dihedral transformations of the original image. They are provided in the following order:

- Vertical flip
- Horizontal flip
- 180 degree rotation
- 90 degree rotation
- 90 degree rotation and vertical flip
- 90 degree rotation and horizontal flip

- 270 degree rotation

Parameters

- **hash_string_or_vector** – The hash string or vector
- **hash_format** – One 'base64' or 'hex'

compute_with_quality(*image*, *hash_format='base64'*)

Compute hash and hash quality from image.

Parameters

- **image** (Union[str, ndarray, Image, BytesIO]) – An image represented as a filepath, a PIL image object, or as an np.ndarray object. If it is an np.ndarray object, it must be in RGB color order (note the OpenCV default is BGR).
- **hash_format** – One 'base64' or 'hex'

Return type Tuple[str, int]

Returns A tuple of (hash, quality)

The following image hash functions are included in the package.

class perception.hashers.image.**AverageHash**(*hash_size=8*)

Computes a simple hash comparing the intensity of each pixel in a resized version of the image to the mean. Implementation based on that of ImageHash.

class perception.hashers.image.**PHash**(*hash_size=8*, *highfreq_factor=4*, *exclude_first_term=False*, *freq_shift=0*)

Also known as the DCT hash, a hash based on discrete cosine transforms of images. See complete paper for details. Implementation based on that of ImageHash.

Parameters

- **hash_size** – The number of DCT elements to retain (the hash length will be hash_size * hash_size).
- **highfreq_factor** – The multiple of the hash size to resize the input image to before computing the DCT.
- **exclude_first_term** – WHether to exclude the first term of the DCT
- **freq_shift** – The number of DCT low frequency elements to skip.

class perception.hashers.image.**WaveletHash**(*hash_size=8*, *image_scale=None*, *mode='haar'*)

Similar to PHash but using wavelets instead of DCT. Implementation based on that of ImageHash.

class perception.hashers.image.**MarrHildreth**

A wrapper around OpenCV's Marr-Hildreth hash. See paper for details.

class perception.hashers.image.**BlockMean**

A wrapper around OpenCV's Block Mean hash. See paper for details.

class perception.hashers.image.**ColorMoment**

A wrapper around OpenCV's Color Moments hash. See paper for details.

class perception.hashers.image.**PDQHash**

The Facebook PDQ hash. Based on the original implementation located at the official repository.

class perception.hashers.image.**DHash**(*hash_size=8*)

A hash based on the differences between adjacent pixels. Implementation based on that of ImageHash.

class perception.hashers.image.**PHashF**(*hash_size=8*, *highfreq_factor=4*, *exclude_first_term=False*, *freq_shift=0*)

A real-valued version of PHash. It returns the raw 32-bit floats in the DCT. For a more compact approach, see PHashU8.

class perception.hashers.image.**PDQHashF**

class perception.hashers.image.**PHashU8**(*hash_size=8*, *highfreq_factor=4*, *exclude_first_term=False*, *freq_shift=0*)

A real-valued version of PHash. It uses minimum / maximum scaling to convert DCT values to unsigned 8-bit integers (more compact than the 32-bit floats used by PHashF at the cost of precision).

Videos

All video hashers inherit from the `VideoHasher` class.

class perception.hashers.hasher.**VideoHasher**

compute(*filepath*, *errors='raise'*, *hash_format='base64'*)

Compute a hash for a video at a given filepath.

Parameters

- **filepath** – Path to video file
- **errors** – One of "raise", "ignore", or "warn". Passed to perception.hashers.tools.read_video.

hash_from_final_state(*state*)

Called after all frames have been processed. Returns the final feature vector.

Parameters **state** (`dict`) – The state dictionary at the end of processing.

Return type `ndarray`

process_frame(*frame*, *frame_index*, *frame_timestamp*, *state=None*)

Called for each frame in the video. For all but the first frame, a state is provided recording the state from the previous frame.

Parameters

- **frame** (`ndarray`) – The current frame as an RGB ndarray
- **frame_index** (`int`) – The current frame index
- **frame_timestamp** (`float`) – The current frame timestamp
- **state** (`Optional[dict]`) – The state from the last call to process_frame

Return type `dict`

The following video hash functions are included in the package.

class perception.hashers.video.**FramewiseHasher**(*frame_hasher*, *interframe_threshold*, *frames_per_second=15*, *quality_threshold=None*)

A hasher that simply returns frame-wise hashes at some regular interval with some minimum inter-frame distance threshold.

compute_batches(*filepath*, *batch_size*, *errors='raise'*, *hash_format='base64'*)

Compute hashes for a video in batches.

Parameters

- **filepath** (str) – Path to video file
- **batch_size** (int) – The batch size to use for returning hashes
- **errors** – One of "raise", "ignore", or "warn". Passed to perception.hashers.tools.read_video.
- **hash_format** – The format in which to return hashes

hash_from_final_state(*state*)
Called after all frames have been processed. Returns the final feature vector.

Parameters **state** – The state dictionary at the end of processing.

process_frame(*frame*, *frame_index*, *frame_timestamp*, *state=None*)
Called for each frame in the video. For all but the first frame, a state is provided recording the state from the previous frame.

Parameters

- **frame** – The current frame as an RGB ndarray
- **frame_index** – The current frame index
- **frame_timestamp** – The current frame timestamp
- **state** – The state from the last call to process_frame

class perception.hashers.video.**TMKL1**(*frame_hasher=None*, *frames_per_second=15*, *dtype='float32'*, *distance_metric='cosine'*, *norm=2*, *quality_threshold=None*)
The TMK L1 video hashing algorithm.

hash_from_final_state(*state*)
Called after all frames have been processed. Returns the final feature vector.

Parameters **state** – The state dictionary at the end of processing.

process_frame(*frame*, *frame_index*, *frame_timestamp*, *state=None*)
Called for each frame in the video. For all but the first frame, a state is provided recording the state from the previous frame.

Parameters

- **frame** – The current frame as an RGB ndarray
- **frame_index** – The current frame index
- **frame_timestamp** – The current frame timestamp
- **state** – The state from the last call to process_frame

class perception.hashers.video.**TMKL2**(*frame_hasher=None*, *frames_per_second=15*, *normalization='matrix'*)
The TMK L2 video hashing algorithm.

hash_from_final_state(*state*)
Called after all frames have been processed. Returns the final feature vector.

Parameters **state** – The state dictionary at the end of processing.

process_frame(*frame*, *frame_index*, *frame_timestamp*, *state=None*)
Called for each frame in the video. For all but the first frame, a state is provided recording the state from the previous frame.

Parameters

- **frame** – The current frame as an RGB ndarray
- **frame_index** – The current frame index
- **frame_timestamp** – The current frame timestamp
- **state** – The state from the last call to process_frame

class perception.hashers.video.**SimpleSceneDetection**(*base_hasher=None*, *interscene_threshold=None*, *min_frame_size=50*, *max_scene_length=None*)

The SimpleSceneDetection hasher is a wrapper around other video hashers to create separate hashes for different scenes / shots in a video. It works by shrinking each frame, blurring it, and doing a simple delta with the previous frame. If they are different, this marks the start of a new scene. In addition, this wrapper will also remove letterboxing from videos by checking for solid black areas on the edges of the frame.

Parameters

- **base_hasher** (Optional[*VideoHasher*]) – The base video hasher to use for each scene.
- **interscene_threshold** – The distance threshold between sequential scenes that new hashes must meet to be included (this is essentially for deduplication)
- **min_frame_size** – The minimum frame size to use for computing hashes. This is relevant for letterbox detection as black frames will tend to be completely "cropped" and make the frame very small.
- **max_scene_length** – The maximum length of a single scene.

compute_batches(*filepath*, *errors='raise'*, *hash_format='base64'*, *batch_size=10*)

Compute a hash for a video at a given filepath and yield hashes in a given batch size.

Parameters

- **filepath** – Path to video file
- **errors** – One of "raise", "ignore", or "warn". Passed to perception.hashers.tools.read_video.
- **hash_format** – The hash format to use when returning hashes.
- **batch_size** – The minimum number of hashes to include in each batch.

hash_from_final_state(*state*)

Called after all frames have been processed. Returns the final feature vector.

Parameters **state** – The state dictionary at the end of processing.

process_frame(*frame*, *frame_index*, *frame_timestamp*, *state=None*, *batch_mode=False*)

Called for each frame in the video. For all but the first frame, a state is provided recording the state from the previous frame.

Parameters

- **frame** – The current frame as an RGB ndarray
- **frame_index** – The current frame index
- **frame_timestamp** – The current frame timestamp
- **state** – The state from the last call to process_frame

Tools

These utility functions are only used by the hashers but are documented here for completeness.

perception.hashers.tools.**compute_md5**(*filepath*)
Compute the md5 hash for a file at *filepath*.

Parameters **filepath** – The path to the file

Return type str

perception.hashers.tools.**compute_quality**(*image*)
Compute a quality metric, using the calculation proposed by Facebook for their PDQ hash algorithm.

perception.hashers.tools.**compute_synchronized_video_hashes**(*filepath*, *hashers*, *framerates=None*, *hash_format='base64'*, *use_queue=True*)
Compute the video hashes for a group of hashers with synchronized frame processing wherever possible.

Parameters

- **filepath** (str) – Path to video file.
- **hashers** (dict) – A dictionary mapping hasher names to video hasher objects
- **hash_format** – The format in which to return the hashes
- **use_queue** – Whether to use queued video frames

perception.hashers.tools.**get_common_framerates**(*id_rates*)
Compute an optimal set of framerates for a list of framerates. Optimal here means that reading the video at each of the framerates will allow one to collect all of the frames required with the smallest possible number of frames decoded.

For example, consider if we need to read a video at 3 fps, 5 fps, 1 fps and 0.5 fps. We could read the video 4 times (once per framerate). But a more optimal approach is to read the video only twice, once at 3 frames per second and another time at 5 frames per second. For the 1 fps hasher, we simply pass every 3rd frame of the 3 fps pass. For the 0.5 fps hasher, we pass every 6th frame of the 3 fps pass. So if you pass this function {A: 3, B: 5, C: 1, D: 0.5}, you will get back {3: [A, C, D], 5: C}.

Parameters **id_rates** (dict) – A dictionary with IDs as keys and frame rates as values.

Returns

A dictionary with framerates as keys and a list of ids as values.

Return type rate_ids

perception.hashers.tools.**get_string_length**(*hash_length*, *dtype*, *hash_format='hex'*)
Compute the expected length of a hash string.

Parameters

- **hash_length** (int) – The length of the hash vector
- **dtype** (str) – The dtype of the vector
- **hash_format** – One of 'base64' or 'hex'

Return type int

Returns The expected string length

perception.hashers.tools.**read**(*filepath_or_buffer*, *timeout=None*)
Read a file into an image object

Parameters

- **filepath_or_buffer** (Union[str, ndarray, Image, BytesIO]) – The path to the file or any object with a *read* method (such as *io.BytesIO*)
- **timeout** – If filepath_or_buffer is a URL, the timeout to use for making the HTTP request.

perception.hashers.tools.**read_video**(*filepath*, *frames_per_second=None*, *max_queue_size=128*, *use_queue=True*, *errors='raise'*)

Provides a generator of RGB frames, frame indexes, and timestamps from a video. This function requires you to have installed ffmpeg.

Parameters

- **filepath** – Path to the video file
- **frames_per_second** (Union[str, float, None]) – How many frames to provide for each second of video. If None, all frames are provided. If frames_per_second is "keyframes", we use ffmpeg to select I frames from the video.
- **max_queue_size** – The maximum number of frames to load in the queue
- **use_queue** – Whether to use a queue of frames during processing
- **errors** – Whether to 'raise', 'warn', or 'ignore' errors

Yields (frame, frame_index, timestamp) tuples

perception.hashers.tools.**string_to_vector**(*hash_string*, *dtype*, *hash_length*, *hash_format*, *verify_length=True*)

Convert hash back to vector.

Parameters

- **hash_string** (str) – The input base64 hash string
- **dtype** (str) – The data type of the hash
- **hash_length** (int) – The length of the hash vector
- **verify_length** (bool) – Whether to verify the string length

perception.hashers.tools.**unletterbox**(*image*)

Obtain bounds on an image that remove the black bars on the top, right, bottom, and left side of an image.

Parameters **image** – The image from which to remove letterboxing.

Return type Optional[Tuple[Tuple[int, int], Tuple[int, int]]]

Returns A pair of coordinates bounds of the form (x1, x2) and (y1, y2) representing the left, right, top, and bottom bounds.

perception.hashers.tools.**vector_to_string**(*vector*, *dtype*, *hash_format*)

Convert vector to hash.

Parameters **vector** (ndarray) – Input vector

2.2.2 Benchmarking

Video Transforms

Transforming videos can be more complex, so we provide the following tools for transforming videos.

2.2.3 Tools

class perception.tools.**SaferMatcher**(*api_key=None*, *username=None*, *password=None*, *url=None*, *hasher=None*, *hasher_api_id=None*, *quality_threshold=90*)

An object for matching hashes with the known CSAM hashes in the Safer matching service. Please contact info@getsafer.io for details on obtaining credentials and information on how match responses are provided.

Here's a minimalist example:

```
from perception import hashers, tools

hasher = hashers.PHash(hash_size=16)
matches = hashers.tools.SaferMatcher(
    api_key='YOUR_API_KEY',
    username='YOUR_USERNAME', # You only need to provide
    password='YOUR_PASSWORD', # an API key OR username/password.
    url='MATCHING_SERVICE_URL'
)
```

For authentication, you must provide the API key OR username and password pair. If neither is provided, the function will attempt to find them as environment variables with names SAFER_MATCHING_SERVICE_API_KEY, SAFER_MATCHING_SERVICE_USERNAME, and SAFER_MATCHING_SERVICE_PASSWORD, respectively. You must also provide the URL endpoint for the matching service, either as a keyword argument or as a SAFER_MATCHING_SERVICE_URL environment variable.

Parameters

- **api_key** (Optional[str]) – A base64 encoded set of matching service credentials
- **username** (Optional[str]) – Matching service username
- **password** (Optional[str]) – Matching service password
- **url** (Optional[str]) – Safer matching service URL
- **hasher** (Optional[*ImageHasher*]) – A hasher to use for matching
- **hasher_api_id** (Optional[str]) – The hasher ID for finding matches.
- **quality_threshold** (int) – The quality threshold filter to use

match(*images*)

Match hashes with the Safer matching service.

Parameters **images** (List[Union[str, Tuple[Union[str, ndarray, Image, BytesIO], str]]]) – A list of image filepaths or (image_like, image_id) tuples.

Return type dict

Returns A dictionary of matches. See Safer matching service documentation (contact Thorn for a copy).

perception.tools.**deduplicate**(*files*, *hashers*, *isometric=False*, *progress=None*)

Find duplicates in a list of files.

Parameters

- **files** (List[str]) – A list of filepaths.
- **hashers** (List[Tuple[*ImageHasher*, float]]) – A list of tuples of the form (hasher, threshold)

- **isometric** (bool) – Whether to compare the rotated versions of the images
- **progress** (Optional[tqdm]) – A tqdm progress indicator

Return type List[Tuple[str, str]]

Returns A list of duplicated file pairs. To use, you can just remove the first entry of each pair from your dataset. The pairs are provided in the event that you wish to apply further analysis.

perception.tools.**deduplicate_hashes**(*hashes*, *threshold*, *hash_format='base64'*, *hasher=None*, *hash_length=None*, *hash_dtype=None*, *distance_metric=None*, *progress=None*)

Find duplicates using a list of precomputed hashes.

Parameters

- **hashes** (List[Tuple[str, Union[str, ndarray]]]) – A list of (id, hash) tuples
- **threshold** (float) – A distance threshold
- **hasher** (Optional[*ImageHasher*]) – A hasher to use for computing distances
- **progress** (Optional[tqdm]) – A tqdm object for reporting progress

Return type List[Tuple[str, str]]

Returns A list of duplicated id pairs. To use, you can just remove the first entry of each pair from your dataset. The pairs are provided in the event that you wish to apply further analysis.

Python Module Index

p

Index