perception

Jun 04, 2021

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
from perception.hashers.image.pdq import PDQHash

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(
```

(continues on next page)

(continued from previous page)

```
        os.path.join('thorn-perceptual-benchmark-v0', '**', '*.jpg')
    )
])

# 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.ImageHasher):
    """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': 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 minimum precision threshold. Here we use 99.99%.
precision_threshold = 99.99

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

metrics_by_transform = hashes.compute_threshold_recall(grouping=['transform_name'],␣
↪precision_threshold=precision_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=[], precision_
↪threshold=precision_threshold).reset_index()
print(tabulate.tabulate(metrics_simple, showindex=False, headers=metrics_simple.
↪columns, tablefmt='rst'))
```

category	transform_name	hasher_name	threshold	recall	precision	n_exemplars
paintings	blur2	ahash	0.0078125	51.724	100	2204
paintings	blur2	blockmean	0.0123967	85.753	100	2204
paintings	blur2	dhash	0.105469	100	100	2204
paintings	blur2	marrhildreth	0.0989583	100	100	2204
paintings	blur2	pdq	0.117188	100	100	2204
paintings	blur2	phash	0.0390625	100	100	2204
paintings	blur2	shrinkhash	60.8112	43.33	100	2204
paintings	blur2	wavelet	0.0117188	66.379	100	2204
paintings	crop0.05	ahash	0.00390625	0.045	100	2204
paintings	crop0.05	blockmean	0.0123967	0.227	100	2204
paintings	crop0.05	dhash	0.210938	7.577	100	2204
paintings	crop0.05	marrhildreth	0.213542	3.584	100	2204
paintings	crop0.05	pdq	0.257812	8.439	100	2204
paintings	crop0.05	phash	0.226562	6.76	100	2204
paintings	crop0.05	shrinkhash	95.0053	2.269	100	2204
paintings	crop0.05	wavelet	0.0078125	0	nan	2204
paintings	gamma2	ahash	0.00390625	0.998	100	2204
paintings	gamma2	blockmean	0.0072314	1.724	100	2204
paintings	gamma2	dhash	0.167969	98.639	100	2204
paintings	gamma2	marrhildreth	0.159722	99.41	100	2204
paintings	gamma2	pdq	0.164062	100	100	2204
paintings	gamma2	phash	0.164062	100	100	2204
paintings	gamma2	shrinkhash	46.5296	0	nan	2204
paintings	gamma2	wavelet	0.0117188	18.512	100	2204
paintings	jpeg95	ahash	0.00390625	4.22	100	2204
paintings	jpeg95	blockmean	0.0134298	28.811	100	2204
paintings	jpeg95	dhash	0.191406	94.782	100	2204

Continued on next page

Table 1 – continued from previous page

category	transform_name	hasher_name	threshold	recall	precision	n_exemplars
paintings	jpeg95	marrhildreth	0.168403	82.985	100	2204
paintings	jpeg95	pdq	0.257812	100	100	2204
paintings	jpeg95	phash	0.234375	100	100	2204
paintings	jpeg95	shrinkhash	66.053	55.172	100	2204
paintings	jpeg95	wavelet	0	0	nan	2204
paintings	noise0.2	ahash	0.00390625	2.677	100	2204
paintings	noise0.2	blockmean	0.00826446	6.987	100	2204
paintings	noise0.2	dhash	0.25	93.648	100	2204
paintings	noise0.2	marrhildreth	0.170139	73.911	100	2204
paintings	noise0.2	pdq	0.257812	99.229	100	2204
paintings	noise0.2	phash	0.257812	100	100	2204
paintings	noise0.2	shrinkhash	169.387	3.312	100	2204
paintings	noise0.2	wavelet	0.0078125	1.407	100	2204
paintings	noop	ahash	0	100	100	2204
paintings	noop	blockmean	0	100	100	2204
paintings	noop	dhash	0	100	100	2204
paintings	noop	marrhildreth	0	100	100	2204
paintings	noop	pdq	0	100	100	2204
paintings	noop	phash	0	100	100	2204
paintings	noop	shrinkhash	0	100	100	2204
paintings	noop	wavelet	0	100	100	2204
paintings	pad0.2	ahash	0.0703125	0	nan	2204
paintings	pad0.2	blockmean	0.0795455	0	nan	2204
paintings	pad0.2	dhash	0.210938	1.089	100	2204
paintings	pad0.2	marrhildreth	0.177083	0	nan	2204
paintings	pad0.2	pdq	0.289062	1.86	100	2204
paintings	pad0.2	phash	0.273438	2.541	100	2204
paintings	pad0.2	shrinkhash	146.325	0.181	100	2204
paintings	pad0.2	wavelet	0.109375	0	nan	2204
paintings	resize0.5	ahash	0.0078125	76.089	100	2204
paintings	resize0.5	blockmean	0.0144628	98.185	100	2204
paintings	resize0.5	dhash	0.0976562	100	100	2204
paintings	resize0.5	marrhildreth	0.154514	99.819	100	2204
paintings	resize0.5	pdq	0.1875	100	100	2204
paintings	resize0.5	phash	0.09375	100	100	2204
paintings	resize0.5	shrinkhash	56.9034	76.27	100	2204
paintings	resize0.5	wavelet	0.0117188	84.71	100	2204
paintings	rotate4	ahash	0.0390625	2.949	100	2204
paintings	rotate4	blockmean	0.0382231	2.949	100	2204
paintings	rotate4	dhash	0.207031	36.298	100	2204
paintings	rotate4	marrhildreth	0.227431	61.978	100	2204
paintings	rotate4	pdq	0.273438	56.08	100	2204
paintings	rotate4	phash	0.257812	61.615	100	2204
paintings	rotate4	shrinkhash	69.1737	2.813	100	2204
paintings	rotate4	wavelet	0.03125	0.136	100	2204
paintings	vignette	ahash	0.0429688	6.171	100	2204
paintings	vignette	blockmean	0.0475207	8.122	100	2204
paintings	vignette	dhash	0.121094	32.305	100	2204
paintings	vignette	marrhildreth	0.177083	77.904	100	2204

Continued on next page

Table 1 – continued from previous page

category	transform_name	hasher_name	threshold	recall	precision	n_exemplars
paintings	vignette	pdq	0.132812	100	100	2204
paintings	vignette	phash	0.132812	100	100	2204
paintings	vignette	shrinkhash	102.186	3.267	100	2204
paintings	vignette	wavelet	0.046875	3.085	100	2204
paintings	watermark	ahash	0.00390625	20.054	100	2204
paintings	watermark	blockmean	0.0123967	45.145	100	2204
paintings	watermark	dhash	0.0585938	100	100	2204
paintings	watermark	marrhildreth	0.0625	100	100	2204
paintings	watermark	pdq	0.273438	98.866	100	2204
paintings	watermark	phash	0.28125	99.456	100	2204
paintings	watermark	shrinkhash	104.398	75.998	100	2204
paintings	watermark	wavelet	0.0117188	51.27	100	2204
photographs	blur2	ahash	0.015625	76.727	100	1650
photographs	blur2	blockmean	0.0330579	98	100	1650
photographs	blur2	dhash	0.0859375	98.97	100	1650
photographs	blur2	marrhildreth	0.107639	97.576	100	1650
photographs	blur2	pdq	0.304688	100	100	1650
photographs	blur2	phash	0.179688	100	100	1650
photographs	blur2	shrinkhash	117.627	44	100	1650
photographs	blur2	wavelet	0.0195312	79.879	100	1650
photographs	crop0.05	ahash	0.0078125	0.182	100	1650
photographs	crop0.05	blockmean	0.0258264	0.788	100	1650
photographs	crop0.05	dhash	0.0976562	1.091	100	1650
photographs	crop0.05	marrhildreth	0.173611	3.152	100	1650
photographs	crop0.05	pdq	0.304688	30.606	100	1650
photographs	crop0.05	phash	0.320312	63.697	100	1650
photographs	crop0.05	shrinkhash	125.94	1.152	100	1650
photographs	crop0.05	wavelet	0.015625	0.182	100	1650
photographs	gamma2	ahash	0.015625	8.182	100	1650
photographs	gamma2	blockmean	0.0268595	17.212	100	1650
photographs	gamma2	dhash	0.101562	90.303	100	1650
photographs	gamma2	marrhildreth	0.105903	90.909	100	1650
photographs	gamma2	pdq	0.210938	100	100	1650
photographs	gamma2	phash	0.234375	100	100	1650
photographs	gamma2	shrinkhash	119.683	0.545	100	1650
photographs	gamma2	wavelet	0.0195312	18.424	100	1650
photographs	jpeg95	ahash	0.0117188	29.879	100	1650
photographs	jpeg95	blockmean	0.0278926	76.788	100	1650
photographs	jpeg95	dhash	0.121094	84.182	100	1650
photographs	jpeg95	marrhildreth	0.104167	69.576	100	1650
photographs	jpeg95	pdq	0.296875	99.879	100	1650
photographs	jpeg95	phash	0.28125	99.879	100	1650
photographs	jpeg95	shrinkhash	131.031	89.212	100	1650
photographs	jpeg95	wavelet	0.0195312	40.242	100	1650
photographs	noise0.2	ahash	0.015625	27.636	100	1650
photographs	noise0.2	blockmean	0.036157	75.091	100	1650
photographs	noise0.2	dhash	0.121094	54.121	100	1650
photographs	noise0.2	marrhildreth	0.0989583	46.364	100	1650
photographs	noise0.2	pdq	0.296875	99.697	100	1650

Continued on next page

Table 1 – continued from previous page

category	transform_name	hasher_name	threshold	recall	precision	n_exemplars
photographs	noise0.2	phash	0.304688	99.818	100	1650
photographs	noise0.2	shrinkhash	210.661	57.576	100	1650
photographs	noise0.2	wavelet	0.0234375	27.03	100	1650
photographs	noop	ahash	0	100	100	1650
photographs	noop	blockmean	0	100	100	1650
photographs	noop	dhash	0	100	100	1650
photographs	noop	marrhildreth	0	100	100	1650
photographs	noop	pdq	0	100	100	1650
photographs	noop	phash	0	100	100	1650
photographs	noop	shrinkhash	0	100	100	1650
photographs	noop	wavelet	0	100	100	1650
photographs	pad0.2	ahash	0.0429688	0.061	100	1650
photographs	pad0.2	blockmean	0.0320248	0	nan	1650
photographs	pad0.2	dhash	0.105469	0.545	100	1650
photographs	pad0.2	marrhildreth	0.177083	0.121	100	1650
photographs	pad0.2	pdq	0.28125	1.455	100	1650
photographs	pad0.2	phash	0.289062	3.515	100	1650
photographs	pad0.2	shrinkhash	114.721	0.061	100	1650
photographs	pad0.2	wavelet	0.0820312	0	nan	1650
photographs	resize0.5	ahash	0.015625	87.697	100	1650
photographs	resize0.5	blockmean	0.0330579	99.152	100	1650
photographs	resize0.5	dhash	0.0898438	98.485	100	1650
photographs	resize0.5	marrhildreth	0.111111	95.394	100	1650
photographs	resize0.5	pdq	0.328125	99.818	100	1650
photographs	resize0.5	phash	0.234375	100	100	1650
photographs	resize0.5	shrinkhash	132.117	80.242	100	1650
photographs	resize0.5	wavelet	0.0195312	88.97	100	1650
photographs	rotate4	ahash	0.0273438	1.818	100	1650
photographs	rotate4	blockmean	0.0371901	3.879	100	1650
photographs	rotate4	dhash	0.09375	2.97	100	1650
photographs	rotate4	marrhildreth	0.149306	4.606	100	1650
photographs	rotate4	pdq	0.304688	73.394	100	1650
photographs	rotate4	phash	0.3125	89.818	100	1650
photographs	rotate4	shrinkhash	130.211	4.424	100	1650
photographs	rotate4	wavelet	0.0078125	0.061	100	1650
photographs	vignette	ahash	0.0273438	8.242	100	1650
photographs	vignette	blockmean	0.0320248	10	100	1650
photographs	vignette	dhash	0.0703125	22	100	1650
photographs	vignette	marrhildreth	0.0954861	38.727	100	1650
photographs	vignette	pdq	0.117188	100	100	1650
photographs	vignette	phash	0.125	100	100	1650
photographs	vignette	shrinkhash	138.989	11.939	100	1650
photographs	vignette	wavelet	0.0195312	4.242	100	1650
photographs	watermark	ahash	0.015625	42.667	100	1650
photographs	watermark	blockmean	0.0247934	60.788	100	1650
photographs	watermark	dhash	0.078125	100	100	1650
photographs	watermark	marrhildreth	0.112847	98.727	100	1650
photographs	watermark	pdq	0.3125	99.818	100	1650
photographs	watermark	phash	0.3125	99.758	100	1650

Continued on next page

Table 1 – continued from previous page

category	transform_name	hasher_name	threshold	recall	precision	n_exemplars
photographs	watermark	shrinkhash	142.046	79.576	100	1650
photographs	watermark	wavelet	0.0195312	53.455	100	1650

transform_name	hasher_name	threshold	recall	precision	n_exemplars
blur2	ahash	0.0078125	49.014	100	3854
blur2	blockmean	0.0123967	80.773	100	3854
blur2	dhash	0.0859375	99.196	100	3854
blur2	marrhildreth	0.107639	98.962	100	3854
blur2	pdq	0.234375	99.948	100	3854
blur2	phash	0.179688	100	100	3854
blur2	shrinkhash	60.8112	28.412	100	3854
blur2	wavelet	0.0117188	62.247	100	3854
crop0.05	ahash	0.00390625	0.052	100	3854
crop0.05	blockmean	0.0123967	0.208	100	3854
crop0.05	dhash	0.0976562	0.493	100	3854
crop0.05	marrhildreth	0.173611	1.635	100	3854
crop0.05	pdq	0.257812	9.03	100	3854
crop0.05	phash	0.226562	7.058	100	3854
crop0.05	shrinkhash	95.0053	1.427	100	3854
crop0.05	wavelet	0.0078125	0	nan	3854
gamma2	ahash	0.00390625	0.934	100	3854
gamma2	blockmean	0.0072314	1.713	100	3854
gamma2	dhash	0.101562	90.036	100	3854
gamma2	marrhildreth	0.105903	94.24	100	3854
gamma2	pdq	0.210938	100	100	3854
gamma2	phash	0.234375	100	100	3854
gamma2	shrinkhash	108.457	0.156	100	3854
gamma2	wavelet	0.0117188	14.997	100	3854
jpeg95	ahash	0.00390625	5.319	100	3854
jpeg95	blockmean	0.0134298	32.045	100	3854
jpeg95	dhash	0.121094	74.079	100	3854
jpeg95	marrhildreth	0.104167	59.263	100	3854
jpeg95	pdq	0.257812	99.896	100	3854
jpeg95	phash	0.234375	99.896	100	3854
jpeg95	shrinkhash	66.053	40.296	100	3854
jpeg95	wavelet	0.00390625	3.71	100	3854
noise0.2	ahash	0.00390625	2.984	100	3854
noise0.2	blockmean	0.00826446	8.563	100	3854
noise0.2	dhash	0.121094	40.088	100	3854
noise0.2	marrhildreth	0.0989583	33.083	100	3854
noise0.2	pdq	0.257812	99.222	100	3854
noise0.2	phash	0.273438	99.896	100	3854
noise0.2	shrinkhash	169.387	4.385	100	3854
noise0.2	wavelet	0.0078125	1.894	100	3854
noop	ahash	0	100	100	3854
noop	blockmean	0	100	100	3854
noop	dhash	0	100	100	3854
noop	marrhildreth	0	100	100	3854

Continued on next page

Table 2 – continued from previous page

transform_name	hasher_name	threshold	recall	precision	n_exemplars
noop	pdq	0	100	100	3854
noop	phash	0	100	100	3854
noop	shrinkhash	0	100	100	3854
noop	wavelet	0	100	100	3854
pad0.2	ahash	0.0429688	0.026	100	3854
pad0.2	blockmean	0.0320248	0	nan	3854
pad0.2	dhash	0.105469	0.234	100	3854
pad0.2	marrhildreth	0.177083	0.052	100	3854
pad0.2	pdq	0.28125	1.349	100	3854
pad0.2	phash	0.273438	2.387	100	3854
pad0.2	shrinkhash	114.721	0.052	100	3854
pad0.2	wavelet	0.0820312	0	nan	3854
resize0.5	ahash	0.0078125	70.784	100	3854
resize0.5	blockmean	0.0144628	95.226	100	3854
resize0.5	dhash	0.0898438	99.299	100	3854
resize0.5	marrhildreth	0.112847	97.846	100	3854
resize0.5	pdq	0.265625	99.844	100	3854
resize0.5	phash	0.234375	100	100	3854
resize0.5	shrinkhash	56.9034	51.453	100	3854
resize0.5	wavelet	0.0117188	80.747	100	3854
rotate4	ahash	0.0273438	1.297	100	3854
rotate4	blockmean	0.0371901	3.036	100	3854
rotate4	dhash	0.09375	1.401	100	3854
rotate4	marrhildreth	0.149306	3.762	100	3854
rotate4	pdq	0.273438	54.489	100	3854
rotate4	phash	0.257812	59.626	100	3854
rotate4	shrinkhash	69.1737	1.894	100	3854
rotate4	wavelet	0.0078125	0.026	100	3854
vignette	ahash	0.0273438	4.67	100	3854
vignette	blockmean	0.0320248	6.098	100	3854
vignette	dhash	0.0703125	12.195	100	3854
vignette	marrhildreth	0.0954861	30.54	100	3854
vignette	pdq	0.132812	100	100	3854
vignette	phash	0.132812	100	100	3854
vignette	shrinkhash	103.005	4.541	100	3854
vignette	wavelet	0.0195312	1.946	100	3854
watermark	ahash	0.00390625	18.5	100	3854
watermark	blockmean	0.0123967	41.593	100	3854
watermark	dhash	0.078125	100	100	3854
watermark	marrhildreth	0.112847	99.455	100	3854
watermark	pdq	0.273438	99.014	100	3854
watermark	phash	0.28125	99.377	100	3854
watermark	shrinkhash	104.398	71.199	100	3854
watermark	wavelet	0.0117188	46.912	100	3854

hasher_name	threshold	recall	precision	n_exemplars
ahash	0.00390625	17.578	100	42394
blockmean	0.00826446	27.714	100	42394
dhash	0.0859375	51.981	99.9952	42394
marrhildreth	0.100694	55.942	99.9957	42394
pdq	0.257812	77.181	99.9969	42394
phash	0.273438	81.967	99.9942	42394
shrinkhash	56.9034	22.378	100	42394
wavelet	0.00390625	18.467	100	42394

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


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.
```

(continues on next page)

(continued from previous page)

```
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
↪'),
    ('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'), ('DVPL2.mp4', 'EI5M3.mp4'), ('0EGNU.mp4', 'CU3JE.mp4
↪'),
    ('94KP4.mp4', '94KP4.mp4'), ('79QDP.mp4', '79QDP.mp4'), ('GKBX9.mp4', 'GKBX9.mp4
↪'),
    ('RX6R8.mp4', 'RX6R8.mp4'), ('PMVT7.mp4', 'PMVT7.mp4'), ('XNXW6.mp4', 'XNXW6.mp4
↪'),
    ('I005F.mp4', 'I005F.mp4'), ('TF95Y.mp4', 'TF95Y.mp4'), ('79QDP.mp4', '79QDP.mp4
↪'),
    ('LQGMM.mp4', 'LQGMM.mp4'), ('QCAUL.mp4', 'QCAUL.mp4'), ('GFVSV.mp4', 'GFVSV.mp4
↪'),
    ('4UYGY.mp4', '4UYGY.mp4'), ('BYDSE.mp4', 'BYDSE.mp4'), ('PV3KQ.mp4', 'PV3KQ.mp4
↪'),
    ('1X0M3.mp4', '1X0M3.mp4'), ('T5FHD.mp4', 'T5FHD.mp4'), ('QRHJJ.mp4', 'QRHJJ.mp4
↪'),
    ('JYBGS.mp4', 'JYBGS.mp4'), ('N2XCF.mp4', 'N2XCF.mp4'), ('OZPA9.mp4', 'OZPA9.mp4
↪'),
```

(continues on next page)

(continued from previous page)

```
    ('297S4.mp4', '297S4.mp4'), ('LHU7D.mp4', 'LHU7D.mp4'), ('TSKZL.mp4', 'TSKZL.mp4
↪'),
    ('BCONW.mp4', 'BCONW.mp4'), ('KBPDM.mp4', 'KBPDM.mp4'), ('7FTBS.mp4', '7FTBS.mp4
↪'),
    ('099Y1.mp4', '099Y1.mp4'), ('S2RIQ.mp4', 'S2RIQ.mp4'), ('22FJU.mp4', '22FJU.mp4
↪'),
    ('99UA6.mp4', '99UA6.mp4'), ('WJ13E.mp4', 'WJ13E.mp4'), ('5OLVC.mp4', '5OLVC.mp4
↪'),
    ('YQ6Z6.mp4', 'YQ6Z6.mp4'), ('T5MLJ.mp4', 'T5MLJ.mp4'), ('0VOQC.mp4', '0VOQC.mp4
↪'),
    ('S2RIQ.mp4', 'S2RIQ.mp4'), ('2VNXF.mp4', '2VNXF.mp4'), ('G87XG.mp4', 'G87XG.mp4
↪'),
    ('RRS54.mp4', 'RRS54.mp4'), ('TXJK7.mp4', 'TXJK7.mp4'), ('G4KE3.mp4', 'G4KE3.mp4
↪'),
    ('3SNSC.mp4', '3SNSC.mp4'), ('U2FA5.mp4', 'U2FA5.mp4'), ('9AFQ7.mp4', '9AFQ7.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',
```

(continues on next page)

(continued from previous page)

```
            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'
        ),
        '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=5)
    # 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(precision_threshold=99.9, grouping=['transform_name'])
```

transform_name	hasher_name	threshold	recall	precision	n_exemplars
black_frames	phashu8_framewise	51.0979	88.12	99.9069	278644
black_frames	phashu8_tmkl1	55.7584	99.918	99.9079	403768
black_padding	phashu8_framewise	74.6391	7.662	100	277399
black_padding	phashu8_tmkl1	53.8702	99.898	99.9079	406899
clip0.2	phashu8_framewise	54.8635	90.741	99.9098	224264
clip0.2	phashu8_tmkl1	59.0424	99.724	99.9077	324251
gif	phashu8_framewise	55.4437	68.21	99.9088	82232
gif	phashu8_tmkl1	55.4887	81.029	99.9103	39757
noop	phashu8_framewise	0	100	100	282658
noop	phashu8_tmkl1	0	100	100	408871
shrink	phashu8_framewise	24.7184	100	100	281731
shrink	phashu8_tmkl1	49.8999	99.836	99.9078	400650
slideshow	phashu8_framewise	56.9825	99.713	99.9076	172829
slideshow	phashu8_tmkl1	56.8683	95.934	99.9035	90684

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'

Return type Optional[str]

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[ndarray, str, None, List[Optional[str]]]

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', 'hex', or 'vector'

Return type Tuple[Union[ndarray, str, None, List[Optional[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.**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.**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'*, ***kwargs*)

Compute a hash for a video at a given filepath. All other arguments are passed to perception.hashers.tools.read_video.

Parameters

- **filepath** – Path to video file
- **errors** – One of "raise", "ignore", or "warn". Passed to perception.hashers.tools.read_video.
- **hash_format** – One of "vector", "base64", or "hex"
- **max_duration** – The maximum length of the video to hash.
- **max_size** – The maximum size of frames to queue

frames_per_second = 1

The frame rate at which videos are read

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** (`Optional[int]`) – The current frame index
- **frame_timestamp** (`Optional[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*, *similarity_threshold=0.95*, *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.
- **similarity_threshold** – The threshold for detecting whether two frames are different enough to constitute a new 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.**b64_to_hex**(*hash_string*, *dtype*, *hash_length*, *verify_length=True*)
Convert a base64-encoded hash to hex.

Parameters

- **hash_string** (str) – The input hex 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.**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.

Return type int

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.**hex_to_b64**(*hash_string*, *dtype*, *hash_length*, *verify_length=True*)

Convert a hex-encoded hash to base64.

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.**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'*, *use_ffmpeg=False*, ***kwargs*)

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

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
- **max_duration** – The maximum length of the video to hash.
- **max_size** – The maximum size of frames to queue
- **errors** – Whether to 'raise', 'warn', or 'ignore' errors
- **use_ffmpeg** – Whether to use the FFMPEG CLI to read videos. If True, other kwargs (e.g., `use_cuda`) are passed to `read_video_to_generator_ffmpeg`.

Yields (frame, frame_index, timestamp) tuples

Return type `Generator[Tuple[ndarray, Optional[int], Optional[float]], None, None]`

`perception.hashers.tools.`**read_video_to_generator**(*filepath*, *frames_per_second=None*, *errors='raise'*, *max_duration=None*, *max_size=None*)

This is used by `read_video` when `use_ffmpeg` is False (default).

Parameters

- **filepath** – See `read_video`.
- **frames_per_second** (`Union[str, float, None]`) – See `read_video`.
- **errors** – See `read_video`.
- **max_duration** (`Optional[float]`) – See `read_video`.
- **max_size** (`Optional[int]`) – See `read_video`.

Return type `Generator[Tuple[ndarray, Optional[int], Optional[float]], None, None]`

Returns See `read_video`.

`perception.hashers.tools.`**read_video_to_generator_ffmpeg**(*filepath*, *frames_per_second=None*, *errors='raise'*, *max_duration=None*, *max_size=None*, *interp=None*, *frame_rounding='up'*, *draw_timestamps=False*, *use_cuda=False*)

This is used by `read_video` when `use_ffmpeg` is True. It differs from `read_video_to_generator` in that it uses FFMPEG instead of OpenCV and, optionally, allows for CUDA acceleration. CUDA acceleration can be faster for larger videos (>1080p) where downsampling is desired. For other videos, CUDA may be slower, but the decoding load will still be taken off the CPU, which may still be advantageous. You can specify which FFMPEG binary to use by setting PERCEPTION_FFMPEG_BINARY.

Parameters

- **filepath** – See read_video
- **frames_per_second** (`Union[str, float, None]`) – See read_video
- **errors** – See read_video
- **max_duration** (`Optional[float]`) – See read_video
- **max_size** (`Optional[int]`) – See read_video

- **interp** (Optional[str]) – The interpolation method to use. When not using CUDA, you must choose one of the interpolation options (default: area). When using CUDA, you must choose from the interp_algo options (default: super).
- **frame_rounding** (str) – The frame rounding method.
- **draw_timestamps** – Draw original timestamps onto the frames (for debugging only)
- **use_cuda** – Whether to enable CUDA acceleration. Requires a CUDA-accelerated version of ffmpeg.

To build FFMPEG with CUDA, do the following in a Docker container based on nvidia/cuda:10.1-cudnn7-devel-ubuntu18.04. The FFMPEG binary will be ffmpeg/ffmpeg.

```
git clone https://git.videolan.org/git/ffmpeg/nv-codec-headers.git
cd nv-codec-headers
make
sudo make install
cd ..
git clone --branch release/4.3 https://git.ffmpeg.org/ffmpeg.git
cd ffmpeg
sudo apt-get update && sudo apt-get -y install yasm
export PATH=$PATH:/usr/local/cuda/bin
./configure --enable-cuda-nvcc --enable-cuvid --enable-nvenc --enable-nvdec
↪              --enable-libnpp --enable-nonfree --extra-cflags=-I/usr/local/cuda/
↪include                    --extra-ldflags=-L/usr/local/cuda/lib64
make -j 10
```

Return type Generator[Tuple[ndarray, Optional[int], Optional[float]], None, None]

Returns See read_video

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

Return type ndarray

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

Return bounds of non-trivial region of image or None.

Unletterboxing is cropping an image such that trivial edge regions are removed. Trivial in this context means that the majority of the values in that row or column are zero or very close to zero. This is why we don't use the terms "non-blank" or "non-empty."

In order to do unletterboxing, this function returns bounds in the form (x1, x2), (y1, y2) where:

- x1 is the index of the first column where over 10% of the pixels have means (average of R, G, B) > 2.
- x2 is the index of the last column where over 10% of the pixels have means > 2.

- y1 is the index of the first row where over 10% of the pixels have means > 2.
- y2 is the index of the last row where over 10% of the pixels have means > 2.

If there are zero columns or zero rows where over 10% of the pixels have means > 2, this function returns *None*.

Note that in the case(s) of a single column and/or row of non-trivial pixels that it is possible for x1 = x2 and/or y1 = y2.

Consider these examples to understand edge cases. Given two images, *L* (entire left and bottom edges are 1, all other pixels 0) and *U* (left, bottom and right edges 1, all other pixels 0), *unletterbox(L)* would return the bounds of the single bottom-left pixel and *unletterbox(U)* would return the bounds of the entire bottom row.

Consider *U1* which is the same as *U* but with the bottom two rows all 1s. *unletterbox(U1)* returns the bounds of the bottom two rows.

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

Return type Optional[str]

2.2.2 Benchmarking

class perception.benchmarking.**BenchmarkImageDataset**(*df*)

categories

The categories included in the dataset

deduplicate(*hasher*, *threshold=0.001*, *isometric=False*)

Remove duplicate files from dataset.

Parameters

- **files** – A list of file paths
- **hasher** (*ImageHasher*) – A hasher to use for finding a duplicate
- **threshold** – The threshold required for a match
- **isometric** – Whether to compute the rotated versions of the images

Return type Tuple[BenchmarkImageDataset, Set[Tuple[str, str]]]

Returns A list where each entry is a list of files that are duplicates of each other. We keep only the last entry.

filter(***kwargs*)

Obtain a new dataset filtered with the given keyword arguments.

classmethod from_tuples(*files*)

Build dataset from a set of files.

Parameters **files** (List[Tuple[str, str]]) – A list of tuples where each entry is a pair filepath and category.

classmethod load(*path_to_zip_or_directory*, *storage_dir=None*, *verify_md5=True*)
Load a dataset from a ZIP file or directory.

Parameters

- **path_to_zip_or_directory** (str) – Pretty self-explanatory
- **storage_dir** (Optional[str]) – If providing a ZIP file, where to extract the contents. If None, contents will be extracted to a folder with the same name as the ZIP file in the same directory as the ZIP file.
- **verify_md5** – Verify md5s when loading

save(*path_to_zip_or_directory*)
Save a dataset to a directory or ZIP file.

Parameters **path_to_zip_or_directory** – Pretty self-explanatory

transform(*transforms*, *storage_dir*, *errors='raise'*)
Prepare files to be used as part of benchmarking run.

Parameters

- **transforms** (Dict[str, Augmenter]) – A dictionary of transformations. The only required key is *noop* which determines how the original, untransformed image is saved. For a true copy, simply make the *noop* key *imgaug.augmenters.Noop()*.
- **storage_dir** (str) – A directory to store all the images along with their transformed counterparts.
- **errors** (str) – How to handle errors reading files. If "raise", exceptions are raised. If "warn", the error is printed as a warning.

Returns A BenchmarkImageTransforms object

Return type transforms

class perception.benchmarking.**BenchmarkImageTransforms**(*df*)

categories
The categories included in the dataset

compute_hashes(*hashers*, *max_workers=5*)
Compute hashes for a series of files given some set of hashers.

Parameters

- **hashers** (Dict[str, *ImageHasher*]) – A dictionary of hashers.
- **max_workers** (int) – Maximum number of workers for parallel hash computation.

Returns A BenchmarkHashes object.

Return type metrics

filter(***kwargs*)
Obtain a new dataset filtered with the given keyword arguments.

classmethod load(*path_to_zip_or_directory*, *storage_dir=None*, *verify_md5=True*)
Load a dataset from a ZIP file or directory.

Parameters

- **path_to_zip_or_directory** (str) – Pretty self-explanatory

- **storage_dir** (Optional[str]) – If providing a ZIP file, where to extract the contents. If None, contents will be extracted to a folder with the same name as the ZIP file in the same directory as the ZIP file.
- **verify_md5** – Verify md5s when loading

save(*path_to_zip_or_directory*)
Save a dataset to a directory or ZIP file.

Parameters **path_to_zip_or_directory** – Pretty self-explanatory

class perception.benchmarking.**BenchmarkVideoDataset**(*df*)

categories
The categories included in the dataset

filter(***kwargs*)
Obtain a new dataset filtered with the given keyword arguments.

classmethod from_tuples(*files*)
Build dataset from a set of files.

Parameters **files** (List[Tuple[str, str]]) – A list of tuples where each entry is a pair filepath and category.

classmethod load(*path_to_zip_or_directory*, *storage_dir=None*, *verify_md5=True*)
Load a dataset from a ZIP file or directory.

Parameters

- **path_to_zip_or_directory** (str) – Pretty self-explanatory
- **storage_dir** (Optional[str]) – If providing a ZIP file, where to extract the contents. If None, contents will be extracted to a folder with the same name as the ZIP file in the same directory as the ZIP file.
- **verify_md5** – Verify md5s when loading

save(*path_to_zip_or_directory*)
Save a dataset to a directory or ZIP file.

Parameters **path_to_zip_or_directory** – Pretty self-explanatory

transform(*transforms*, *storage_dir*, *errors='raise'*)
Prepare files to be used as part of benchmarking run.

Parameters

- **transforms** (Dict[str, Callable]) – A dictionary of transformations. The only required key is *noop* which determines how the original, untransformed video is saved. Each transform should be a callable function with that accepts an *input_filepath* and *output_filepath* argument and it should return the *output_filepath* (which may have a different extension appended by the transform function).
- **storage_dir** (str) – A directory to store all the videos along with their transformed counterparts.
- **errors** (str) – How to handle errors reading files. If "raise", exceptions are raised. If "warn", the error is printed as a warning.

Returns A BenchmarkVideoTransforms object

Return type transforms

class perception.benchmarking.**BenchmarkVideoTransforms**(*df*)

categories
The categories included in the dataset

compute_hashes(*hashers*, *max_workers=5*)
Compute hashes for a series of files given some set of hashers.

Parameters

- **hashers** (Dict[str, *VideoHasher*]) – A dictionary of hashers.
- **max_workers** (int) – Maximum number of workers for parallel hash computation.

Returns A BenchmarkHashes object.

Return type hashes

filter(***kwargs*)
Obtain a new dataset filtered with the given keyword arguments.

classmethod load(*path_to_zip_or_directory*, *storage_dir=None*, *verify_md5=True*)
Load a dataset from a ZIP file or directory.

Parameters

- **path_to_zip_or_directory** (str) – Pretty self-explanatory
- **storage_dir** (Optional[str]) – If providing a ZIP file, where to extract the contents. If None, contents will be extracted to a folder with the same name as the ZIP file in the same directory as the ZIP file.
- **verify_md5** – Verify md5s when loading

save(*path_to_zip_or_directory*)
Save a dataset to a directory or ZIP file.

Parameters **path_to_zip_or_directory** – Pretty self-explanatory

class perception.benchmarking.**BenchmarkHashes**(*df*)
A dataset of hashes for transformed images. It is essentially a wrapper around a *pandas.DataFrame* with the following columns:

- guid
- error
- filepath
- category
- transform_name
- hasher_name
- hasher_dtype
- hasher_distance_metric
- hasher_hash_length
- hash

categories
The categories included in the dataset

compute_threshold_recall(*precision_threshold=99.9*, *grouping=None*, ***kwargs*)
Compute a table for threshold and recall for each category, hasher, and transformation combinations. Additional arguments passed to compute_metrics.

Parameters

- **precision_threshold** – The precision threshold to use for choosing a distance threshold for each hasher.
- **grouping** – List of fields to group by. By default, all fields are used (category, and transform_name).

Return type DataFrame

Returns A pandas DataFrame with 7 columns. The key columns are threshold (The optimal distance threshold for detecting a match for this combination), recall (the number of correct matches divided by the number of possible matches), and precision (the number correct matches divided by the total number of matches whether correct or incorrect).

filter(***kwargs*)
Obtain a new dataset filtered with the given keyword arguments.

show_histograms(*grouping=None*, *precision_threshold=99.9*, ***kwargs*)
Plot histograms for true and false positives, similar to https://tech.okcupid.com/evaluating-perceptual-image-hashes-okcupid/ Additional arguments passed to compute_metrics.

Parameters **grouping** – List of fields to group by. By default, all fields are used (category, and transform_name).

Video Transforms

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

perception.benchmarking.video_transforms.**get_simple_transform**(*width=-1*, *height=-1*, *pad=None*, *codec=None*, *clip_pct=None*, *clip_s=None*, *sar=None*, *fps=None*, *output_ext=None*)
Resize to a specific size and re-encode.

Parameters

- **width** (Union[str, int]) – The target width (-1 to maintain aspect ratio)
- **height** (Union[str, int]) – The target height (-1 to maintain aspect ratio)
- **pad** (Optional[str]) – An ffmpeg pad argument provided as a string.
- **codec** (Optional[str]) – The codec for encoding the video.
- **fps** – The new frame rate for the video.
- **clip_pct** (Optional[Tuple[float, float]]) – The video start and end in percentages of video duration.
- **clip_s** (Optional[Tuple[float, float]]) – The video start and end in seconds (used over clip_pct if both are provided).

- **sar** – Whether to make all videos have a common sample aspect ratio (i.e., for all square pixels, set this to '1/1').
- **output_ext** – The extension to use when re-encoding (used to select video format). It should include the leading '.'.

perception.benchmarking.video_transforms.**get_black_frame_padding_transform**(*duration_s=0*, *duration_pct=0*)

Get a transform that adds black frames at the start and end of a video.

Parameters

- **duration_s** – The duration of the black frames in seconds.
- **duration_pct** – The duration of the black frames as a percentage of video duration. If both duration_s and duration_pct are provided, the maximum value is used.

perception.benchmarking.video_transforms.**get_slideshow_transform**(*frame_input_rate*, *frame_output_rate*, *max_frames=None*, *offset=0*)

Get a slideshow transform to create slideshows from videos.

Parameters

- **frame_input_rate** – The rate at which frames will be sampled from the source video (e.g., a rate of 1 means we collect one frame per second of the input video).
- **frame_output_rate** – The rate at which the sampled frames are played in the slideshow (e.g., a rate of 0.5 means each frame will appear for 2 seconds).
- **max_frames** – The maximum number of frames to write.
- **offset** – The number of seconds to wait before beginning the slide show.

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.

2.2.4 Experimental

This module contains experimental functionality that may not be ready for production use.

Approximate Nearest Neighbors

class perception.experimental.ann.**ApproximateNearestNeighbors**(*con*, *table*, *paramstyle*, *index*, *hash_length*, *metadata_columns=None*, *dtype='uint8'*, *distance_metric='euclidean'*)

A wrapper for a FAISS index.

Parameters

- **con** – A database connection from which to obtain metadata for matched hashes.
- **table** – The table in the database that we should query for metadata.
- **paramstyle** – The parameter style for the given database
- **index** – A FAISS index (or filepath to a FAISS index)
- **hash_length** – The length of the hash that is being matched against.
- **metadata_columns** – The metadata that should be returned for queries.
- **dtype** – The data type for the vectors
- **distance_metric** – The distance metric for the vectors

classmethod from_database(*con*, *table*, *paramstyle*, *hash_length*, *ids_train=None*, *train_size=None*, *chunksize=100000*, *metadata_columns=None*, *index=None*, *gpu=False*, *dtype='uint8'*, *distance_metric='euclidean'*)

Train and build a FAISS index from a database connection.

Parameters

- **con** – A database connection from which to obtain metadata for matched hashes.
- **table** – The table in the database that we should query for metadata.
- **paramstyle** – The parameter style for the given database
- **hash_length** – The length of the hash that is being matched against.
- **ids_train** – The IDs for the vectors to train on.
- **train_size** – The number of vectors to use for training. Will be randomly selected from 1 to the number of vectors in the database. Ignored if ids_train is not None.
- **chunksize** – The chunks of data to draw from the database at a time when adding vectors to the index.
- **metadata_columns** – The metadata that should be returned for queries.
- **index** – If a pretrained index is provided, training will be skipped, any existing vectors will be discarded, and the index will be repopulated with the current contents of the database.

- **gpu** – If true, will attempt to carry out training on a GPU.
- **dtype** – The data type for the vectors
- **distance_metric** – The distance metric for the vectors

nlist
The number of lists in the index.

nprobe
The current value of nprobe.

ntotal
The number of vectors in the index.

query_by_id(*ids*, *include_metadata=True*, *include_hashes=False*)
Get data from the database.

Parameters

- **ids** – The hash IDs to get from the database.
- **include_metadata** – Whether to include metadata columns.
- **include_hashes** – Whether to include the hashes

Return type `DataFrame`

save(*filepath*)
Save an index to disk.

Parameters **filepath** – Where to save the index.

search(*queries*, *threshold=None*, *threshold_func=None*, *hash_format='base64'*, *k=1*)
Search the index and return matches.

Parameters

- **queries** (`List[QueryInput]`) – A list of queries in the form of {"id": <id>, "hash": "<hash_string>"}
- **threshold** (`Optional[int]`) – The threshold to use for matching. Takes precedence over threshold_func.
- **threshold_func** (`Optional[Callable[[ndarray], int]]`) – A function that, given a query vector, returns the desired match threshold for that query.
- **hash_format** – The hash format used for the strings in the query.
- **k** – The number of nearest neighbors to return.

Returns

{ "id": <query ID>, "matches": [{"distance": <distance>, "id": <match ID>, "metadata": {}}]}

The metadata consists of the contents of the metadata columns specified for this matching instance.

Return type Matches in the form of a list of dicts of the form

set_nprobe(*nprobe*)
Set the value of nprobe.

Parameters **nprobe** – The new value for nprobe

Return type `int`

string_to_vector(*s*, *hash_format='base64'*)
Convert a string to vector form.

Parameters

- **s** (str) – The hash string
- **hash_format** – The format for the hash string

Return type ndarray

tune(*n_query=100*, *min_recall=99*, *max_noise=3*)
Obtain minimum value for nprobe that achieves a target level of recall. :param n_query: The number of hashes to use as test hashes. :param min_recall: The minimum desired recall for the index. :param max_noise: The maximum amount of noise to add to each test hash

Returns A tuple of recall, latency (in ms), and nprobe where the nprobe value is the one that achieved the resulting recall.

Raises TuningFailure if no suitable nprobe value is found.

vector_to_string(*vector*, *hash_format='base64'*)
Convert a vector back to string

Parameters

- **vector** – The hash vector
- **hash_format** – The format for the hash

Return type Optional[str]

perception.experimental.ann.**serve**(*index*, *default_threshold=None*, *default_threshold_func=None*, *default_k=1*, *concurrency=2*, *log_level=20*, *host='localhost'*, *port=8080*)
Serve an index as a web API. This function does not block. If you wish to use the function in a blocking manner, you can do something like

```
loop = asyncio.get_event_loop()
loop.run_until_complete(serve(...))
loop.run_forever()
```

You can query the API with something like:

```
curl --header "Content-Type: application/json" \
     --request POST \
     --data '{"queries": [{"hash": "<hash string>", "id": "bar"}], "threshold":␣
↪1200}' \
     http://localhost:8080/v1/similarity
```

Parameters

- **index** (ApproximateNearestNeighbors) – The underlying index
- **default_threshold** (Optional[int]) – The default threshold for matches
- **default_k** (int) – The default number of nearest neighbors to look for
- **concurrency** (int) – The number of concurrent requests served
- **log_level** – The log level to use for the logger
- **host** – The host for the servoce
- **port** – The port for the service

Local Descriptor Deduplication

perception.experimental.local_descriptor_deduplication.**deduplicate**(*filepaths*, *max_features=256*, *min_features=10*, *max_size=256*, *coarse_pct_probe=0*, *coarse_threshold=100*, *minimum_coarse_overlap=0.01*, *minimum_validation_match=0.4*, *minimum_validation_intersection=0.6*, *minimum_validation_inliers=5*, *ratio=0.5*, *max_workers=None*, *use_gpu=True*)

Deduplicate images by doing the following:

1. Unletterbox all images and resize to some maximum size, preserving aspect ratio.
2. Compute the SIFT descriptors and keypoints for all the resulting images.
3. Perform a coarse, approximate search for images with common features.
4. For each candidate pair, validate it pairwise by checking the features and keypoints with the traditional approach using the ratio test. See validate_match for more information.

Parameters

- **filepaths** (Iterable[str]) – The list of images to deduplicate.
- **max_features** (int) – The maximum number of features to extract.
- **min_features** (int) – The minimum number of features to extract.
- **max_size** (int) – The maximum side length for an image.
- **coarse_pct_probe** (float) – The minimum fraction of nearest lists to search. If the product of pct_probe and the number of lists is less than 1, one list will be searched.
- **corase_threshold** – The threshold for a match as a euclidean distance.
- **minimum_coarse_overlap** (float) – The minimum overlap between two files to qualify as a match.
- **minimum_validation_match** (float) – The minimum number of matches passing the ratio test.
- **minimum_validation_intersection** (float) – The minimum overlapping area between the keypoints in the filtered set of matches and the original keypoints.
- **minimum_validation_inliers** (int) – The minimum number of inliers for the transformation matrix.
- **ratio** (float) – The ratio to use for Lowe's ratio test.
- **max_workers** (Optional[int]) – The maximum number of threads to use for doing the final validation step.

Return type List[Tuple[str, str]]

Returns A list of pairs of file duplicates.

perception.experimental.local_descriptor_deduplication.**validate_match**(*kp1*, *des1*, *kp2*, *des2*, *dims1*, *dims2*, *minimum_match=0.4*, *minimum_intersection=0.6*, *minimum_inliers=5*, *ratio=0.5*)

Validate the match between two sets of keypoints and descriptors. The validation algorithm is as follows:

1. Compute the mutual set of matches between the two sets of descriptors and filter them using Lowe's ratio test.
2. If the minimum number of passing matches is less than "minimum_match", the match fails. This ensures we don't have trivial matches.
3. Compute the intersection area of the matched keypoints versus the raw keypoints. If the area overlap is less than minimum_intersection, the match fails. This ensures we don't match on small subsegments of an image, such as logos.
4. Compute a transformation matrix using cv2.findHomography. If we cannot obtain a transformation matrix, the match fails. If the sum total of inliers for the transformation matrix is less than minimum_inliers, the match fails.
5. Finally, use the transformation matrix on a set of points representing the bounding box of each image. If less than minimum_intersection of the bounding box fits within the bounds of the transformed version, the match fails. This is a second pass safety check for logos and other subsegments of images.

Parameters

- **kp1** (ndarray) – The first set of keypoints
- **des1** (ndarray) – The first set of descriptors
- **kp2** (ndarray) – The second set of keypoints
- **des2** (ndarray) – The second set of descriptors
- **dims1** (Tuple[int, int]) – The dimensions (width, height) for the first image
- **dims2** (Tuple[int, int]) – The dimensions (width, height) for the second image
- **minimum_match** (float) – The minimum number of matches passing the ratio test.
- **minimum_intersection** (float) – The minimum overlapping area between the keypoints in the filtered set of matches and the original keypoints.
- **minimum_inliers** (int) – The minimum number of inliers for the transformation matrix.
- **ratio** – The ratio to use for Lowe's ratio test.

Return type float

Returns True if the match passes, False otherwise.

Python Module Index

p

Index

Q

R

S

T

U

V

W