Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 225 (2023) 3143–3152

Procedia Computer Science

www.elsevier.com/locate/procedia

27th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2023)

Multispectral Texture Benchmark

Pavel Kříž[a], Michal Haindl[b,*]

[a] *Faculty of Information Technology, Czech Technical University, Department of Software Engineering, Thákurova 9, 160 00 Prague, Czechia*

[b] *Institute of Information Theory and Automation, Czech Academy of Sciences, Pattern Recognition Department, Pod Vodárenskou věží4, 182 08 Prague, Czechia*

Abstract

Dozens of textural features have been published, but their realistic validation for efficient recognition applications still needs to be discovered. Textural features are derived using various approaches. We present a benchmark that can be used to evaluate these features and categorize them based on their information efficiency. We propose how the features can be benchmarked and explain different ways of measuring their properties and performance. Most textural feature-extracting algorithms are only based on information extraction from monospectral images (gray-level). Apart from native multispectral algorithms, we generalize some of these originally monospectral features for hyperspectral textures in our illustrating examples.

Keywords: Textural features; benchmark; representation; multispectral features.

1. Introduction

Visual scene recognition is predominantly based on visual textures representing an object's material properties. Textural features are a building block for such material recognition, and different features are well-suited for different applications. However, there need to be more studies comparing possible textural features that often differ in their environmental factors, image datasets, hardware, and other variables. We propose a tool to benchmark textural features, compare them, and analyze them. We consider unlimited image spectra in our *textural benchmark*. Previous publications mainly consider textural features extracted from monospectral images and thus ignore all available information. This potentially lost information inevitably and pointlessly lowers the probability of correct material recognition.

* Corresponding author. Tel.: +420-26605-2350 ; fax: +0-000-000-0000.
E-mail address: haindl@utia.cas.cz

10.1016/j.procs.2023.10.308

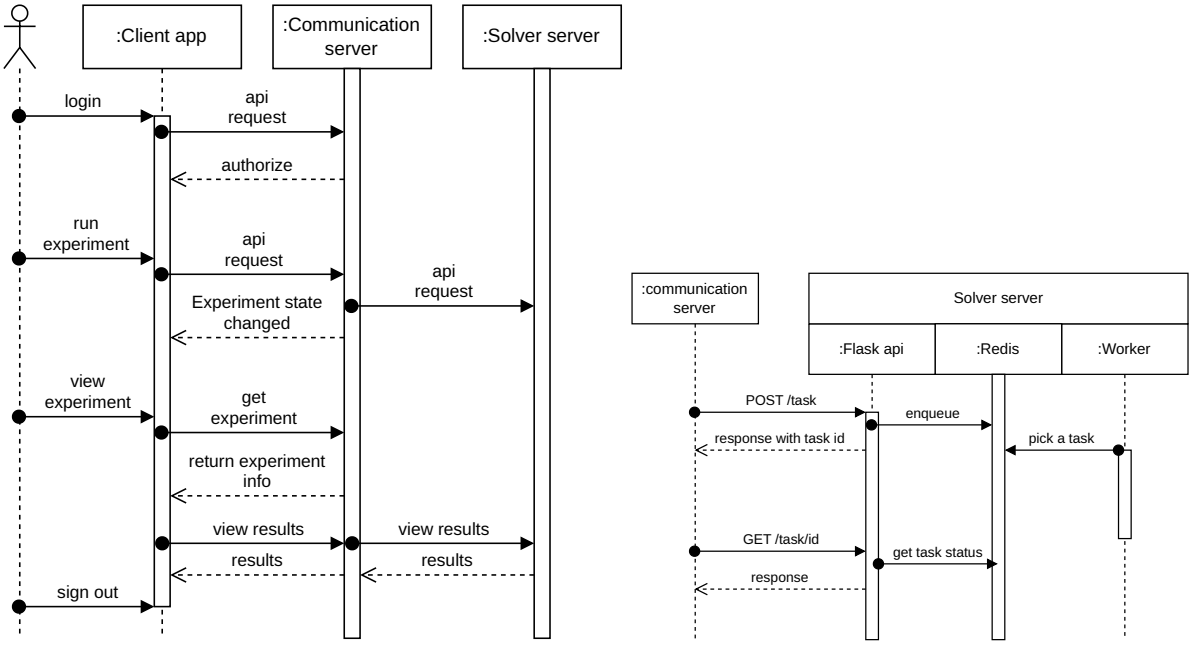

Fig. 1. Sequence diagram describing the communication between the services and client application (left) and the sequence diagram describing the communication inside of the solver service (right).

1.1. Benchmark

The main objective is to develop an online application to run textural feature-extracting algorithms. We have decided to develop a benchmark similar to existing benchmarking applications with the most significant focus on The *Prague Texture Segmentation Data Generator and Benchmark [17] (Mosaic)*. The *Mosaic* benchmark aims to test solutions to the image segmentation problem. Solving that would include feature extraction and classification. Our benchmark focuses only on textural feature evaluation, it does feature extraction as well, but this process is not the subject of evaluation, only the resulting textural information. The *Mosaic* benchmark [17] inspired our presented benchmark, including user registration, comparison of others' results, and input of the prepared dataset. The main goal of the intended user is to obtain textural feature performance statistics in order to be able to compare different features. In addition, the proposed benchmark simultaneously considers all spectra of images. We have aimed to develop the benchmark with that in mind. Even though the current dataset contains only color (three spectral images RGB), it allows working with hyperspectral images having a variable number of spectra.

2. Textural Benchmark

The implemented application consists of three main parts. One part handles communication, one feature extraction and evaluation, and the last one control. The first two are running on a server, and the user can access the benchmark through the client application, the third part, or via an API provided by the first part meant for communication Fig. 8. Another reason we have chosen to split the service into multiple parts is easier system maintenance and, eventually, scaling the benchmark performance. The last property is essential because feature extraction or selected feature evaluation is computationally expensive. The example diagram Fig. 1-left shows the user interacting with the frontend client application, which communicates to the backend (server) in two parts. The first part is the main server that serves the client application and distributes the task to the second part, the solver server. The solver server runs the algorithms and all the managing around that.

2.1. Client Application

We have chosen *Flutter* for its flexibility, allowing us to create prototypes and develop them further into production-ready applications. We considered that a significant advantage since we want benchmark development flexibility for future requirements.

2.2. Communication Server

We chose *Python* because of its suitability in another part of the application. The application architecture is built so that the individual parts communicate using HTTP APIs. For that, we have chosen *Flask* because of its popularity, support, and the fact that it is a lightweight web framework compared to other ones. We have seen the last one as an advantage because we only wanted to create web APIs for communication and keep that part of the codebase simple.

2.3. Solver Server

For the solver server responsible for extracting features and gathering feature statistics, we decided on *Python* because there are many libraries for computer vision and machine learning in *Python*. The *Redis Queue* is a good fit for its simplicity, making it easy to use. It does not offer advanced features but has enough abilities for this benchmark.

The solver server contains three functional objects. First is an API that serves the incoming requests, and if there is a computationally demanding request, it enqueues that requested task into a queue. The queue is a second part implemented using the *Redis Queue*, a library for enqueuing tasks. The *Redis Queue* is a simple queue with enqueued tasks implemented using the Redis key-value database engine. The worker, the third part, is a process that picks up tasks from the queue and solves them. The diagram Fig. 1-right illustrates this process. This distribution is advantageous in the case of scaling, in which it is only needed to create more workers, which can be quickly done.

2.4. User Interface

This section describes the user interface and its testing. The user interface is vital for a good understanding of the application and a good evaluation of the experience of the application by the user. We have created mockups and tested them with the cognitive walkthrough method with the users. With the testing, we have come up with the following interface.

The application allows users to manage and save their experiments. Users can add and open experiments from *My experiments* window, as shown in Fig. 2, a screen the user first sees when logging in to the application. To share experiments with others, the user can publish experiments and then view the experiment in *Public space*, a place where the user can look at published experiments of others, as shown in Fig. 3.

3. Benchmark Features Categorization

Multiple features benchmarking is one significant problem. Textural features differ and have their specific parameters to be selected. We want to compare various features and measure their critical properties. In order to make this measurement general, we have categorized features into several representative groups. This allows us to significantly reduce the code implementation and make the benchmark able to evaluate future features yet to be included.

3.1. Examined Features

We have examined various features to find a generalized form for benchmarking and managing numerous variable textural features. This task is primarily crucial in order to be able to measure their precision. The benchmark recently contains it is prepared for the following features: *LBP family (LBP, LBPRI, MBP, CBP, CLBP, DLBP, LBP_HF); HOG [5]; Gabor features [6]; ACF [13]; LAWS [14]; Tamura features; Haralick features[11]; Textons; VS-LWIR [1]; ORB [22]; SIFT; SURF [3]; FAST [21]; BRIEF [4]; GCM [12]; Markovian features [20]*. We have examined how these features could be represented in generalized form in order also to be able to work with them. We divided them into two main computational division groups: *per-pixel* and *per-window* features.

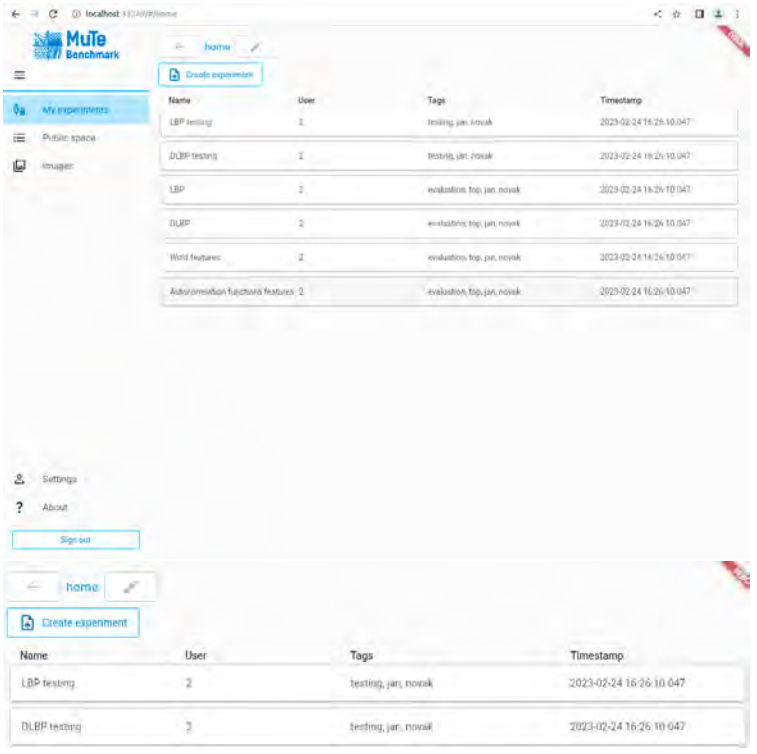

Fig. 2. Users' experiments are on top, and detail is on the bottom.

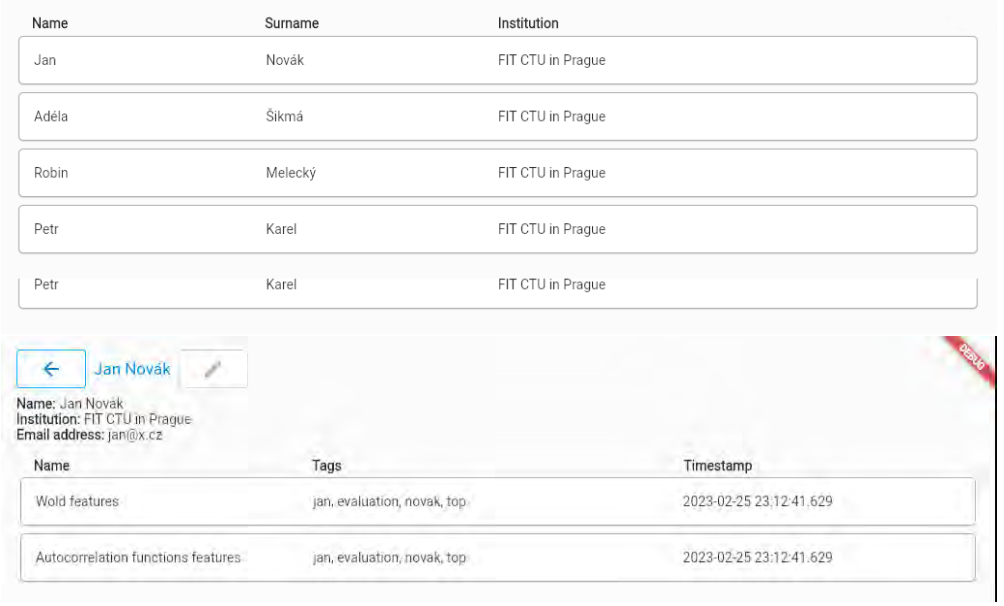

Fig. 3. The public space window is on top, and published experiments of the virtual user *Jan Novák* are on the bottom.

3.2. Per-pixel Features

We call *per-pixel* features calculated separately for every pixel. They usually represent a local relationship of an image pixel in its neighborhood. A good example is the derivative value of an image at each pixel location. Such features have mostly low computation complexity, which is advantageous for real-time image segmentation. In the case of dichotomic or multiclass classification, classifying pixels or local areas is possible without any further calculation. Another approach is the whole image processing, which is needed for histogram construction. The whole image processing is needed because the dimensionality of the histogram features is usually too high for reliable learning.

3.3. Per-window Features

Per-window features are calculated for the whole image window, and their overall dimensionality and size are usually larger than the image size. Unlike *per-pixel* features, Per-window features do not encode small local relationships between pixels, making them unsuitable for precise segmentation regional border localization.

3.4. Representation Sub-division

For the *per-pixel* features, we distinguish only *vector*, for the *per-image* features, where we divide them into the following forms: *vector* (we distinguish *histogram* as a particular case of a *vector*), *list* and *Big matrix*. They all (*per-pixel vector*) require unique metadata to describe their internal structure.

Vector represents an ordered set of numbers, where numbers do not have a special meaning (one number vector of LBP), but in some cases, these values have a special meaning (Tamura features have different statistics in the Vector), then there has to be second Vector storing information about each vector field. *Histogram* is a particular case of a vector format, where the numbers represent the frequency of occurrence of certain features. Visualizing Histogram in a graph makes sense, unlike a general *vector*.

List is a vector that contains sub-vectors as elements. Each element may have in addition, coordinates (x, y) indicating their original place of extraction in the image.

Big matrix format is a multi-dimensional matrix that can contain several sub-matrices, allowing for storing complex data structures such as Markovian features, which have equation parameters represented as matrices.

In our representation, the researched features can be classified into the following categories:

Per-pixel

- **Vector:** LBP (LBPri, MBP, CBP, CLBP, DLBP), HOG (orientations only), Gabor, ACF, LAWS

Per-window

- **Vector:** Tamura, Haralick
- **Histogram:** HOG, Textons, LBP_HF
- **List:** ORB, SIFT, SURF, FAST, BRIEF, VS-LWIR
- **Big matrix:** GCM, Markovian features

Note: We can transform *per-pixel* features to represent the image by using some transformation, such as constructing a histogram. This way, we will get *per-image vector* or *per-window histogram*. However, the process is more complicated when making *per-window* features represent pixels or areas of the image. We propose running the feature extraction algorithm on a given area (or block around a pixel) and then storing it independently for that area. This way, we can achieve similar properties to *per-pixel* features, but the fine representation form remains the same, i.e., *List* is not converted to a vector but remains a *List*. The same goes for the rest (*Vector, Histogram, Big matrix*).

Fig. 4. On the left is the input image, and on the right are extracted $LBP^{ri}_{8,3}$ features.

3.5. Textural Features Examples

Numerous textural features exist from which we can choose. In this section, we will list a few of them, and we will provide examples of the features that have been extracted. The form of the extracted features will be used in later sections to categorize different types of features.

Local Binary Patterns

Local Binary Patterns is a feature extraction method that is a popular option with many variants: LBP [19]; MBP [9]; CBP [7]; CLBP [8]; LBP_HF [2]; $LBP^{ri}_{P}R$, $LBP^{riu}_{P}R$ [18]; DLBP [15]. The principle is to compute features for every pixel and then create. The features are numbers, and every one of them represents some specific micropattern. LBP features are calculated *per-pixel*, and there is one value ranging from 0 to 255 for each pixel. These features are shown in Fig. 4, where we have visualized the $LBP^{ri}_{8,3}$ features extracted from the input image on the left.

Tamura Features

In the original paper, Tamura Features are described as "*Textural features corresponding to human visual perception are useful for optimum feature selection and texture analyzer design. We approximated six basic textural features in the computational form: coarseness, contrast, directionality, line-likeness, regularity, and roughness.*" [23]. Tamura features are calculated *per-window*. Their representations may look as follows:

Listing 1. Example of Tamura features saved in JSON file

```
{
    "coarseness" : 10.5,
    "contrast" : 517.3,
    "directionality" : 0.92,
    "line-likeness" : 0.73,
    "roughness" : -243.12,
    "regularity" : 229.7
}
```

Textons

Textons introduced in [24] represent an image's responses to a set of filters. The histogram size is usually in the magnitude of dozens. Textons features are calculated *per-window* and are composed of defined statistics. Their representations may look as seen in Fig. 5.

Local Scale-Invariant Features

Local Scale-Invariant Features (SIFT) are popular features, and one of their use cases is object matching in an image [16]. SIFT is detected in informative places of an image called keypoints. They consist of image space coordinates

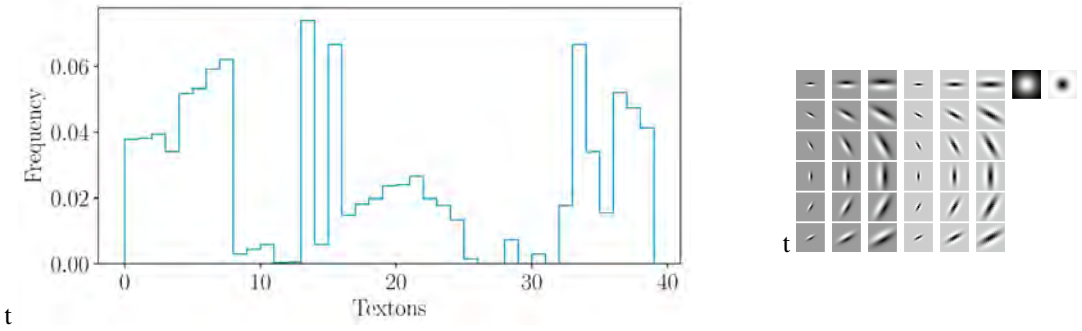

Fig. 5. Visualisation of Textons features in form of a histogram (left) and the MR8 texture bank (right).

Listing 2. Example of SIFT features saved in JSON file

```
{[
  { "keypoint": [25, 123],
    "descriptor": [0.03, 0.0065,...126 numbers]
  },
  { "keypoint": [185, 64],
    "descriptor":[0.0013, 0.0126,...126 numbers]
  }
]}
```

Grey scale	Multispectral
GLCM based	GCM based
Tamura features	Markovian features
Local Binary Pattern	

Table 1. Examples of multispectral and grey scale features [10]. GLCM stands for gray-level co-occurrence matrix, and GCM stands for generalized co-occurrence matrix.

and a description of the surrounding area as a vector with 128 numbers. SIFT is calculated *per-window*, and they are composed of a list of keypoints and their descriptors, which capture the local appearance of the image. The representations of SIFT features may take the form of a list of keypoints and their descriptors, as shown in listing 2.

3.6. Multispectral and Monospectral Features

Another important division is whether the features are natively multispectral or contain only information extracted from a single spectrum. Most of the popular features are monospectral. Usually, they are extracted from grayscale images [10]. Such features can be extracted from all image spectra if they are extracted from each spectrum independently. Tab. 1 shows a few examples of discussed spectral division.

4. Feature Evaluation

We have selected for use in our benchmark several criteria for feature evaluation. We aim to measure the properties of the features but face several obstacles. Firstly, it is not easy to measure computation speed since the hardware used can affect the results, even when using the same algorithm. Moreover, we cannot guarantee that our implementation of feature extraction, or that of others, is optimal. Secondly, some statistics need to indicate whether the result is good. For instance, when considering entropy, lower entropy in features than image entropy suggests that the data is more structured, which is generally desirable. However, this is not a strong indication, and we cannot make definitive statements based solely on such results. Nevertheless, we can still compute some statistics to gain insights into the

properties of the features. We propose the following statistics to be measured: *entropy, mutual information, size, Pearson correlation*.

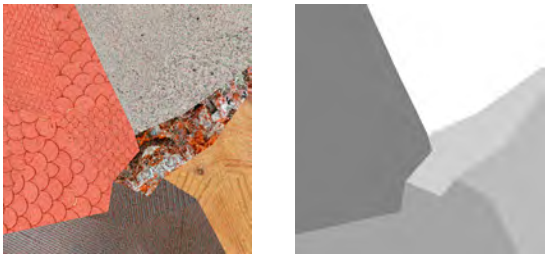

Fig. 6. Segmented image on the left and ground truth for this task on the right. Source: Mosaic portal (`mosaic.utia.cas.cz/`)

4.1. (Un)Supervised Classification

Besides statistics, we propose using simple classification to verify feature invariance to some image transformations, namely scaling, rotation, change of illumination, and degradation (noise). To prepare the input, we load a dataset of dozens of images and test the invariancy of features to each transformation. First, we split all images into two halves. We use the first half as the training set and the second as the test set. The classifier is trained by extracting features from all the first halves of the images. If the features are *per-pixel*, the histogram is created from those and the histogram is then used for classification as a class representation (each image represents a class). The benchmarking process is divided into rounds for every image and transformation. For testing each transformation, there are as many rounds as there are images; in each round, one-half of an image is transformed and classified. The invariance of the features of this transformation is then measured as a percentage of the classification success rate.

Similar to the previously explained classification, a segmentation task can achieve similar results. A similar approach is taken in an online benchmarking portal *Mosaic* [17], which focuses on classification accuracy in general but also offers tools to create image datasets with textured segments that can be transformed using various transformations, as shown in Figure 6, which displays an image with ground truth, where the red texture clearly shows a combination of rotation and scale transformation.

5. Benchmark Processing Pipeline

All of the previously mentioned methodologies are connected to create a web-based benchmark. The diagram in Fig. 8 illustrates the process. The images are first prepared as shown in Fig. 7. The first preparation step is to split the image into spectral planes (image planes) and then into halves. Native multispectral features omit this step. As seen in the process in Fig. 8, the features are extracted, and then the statistics are obtained from the extracted features. The measure of invariance (invariance to image transformations like scaling or rotating) is obtained in the classification process.

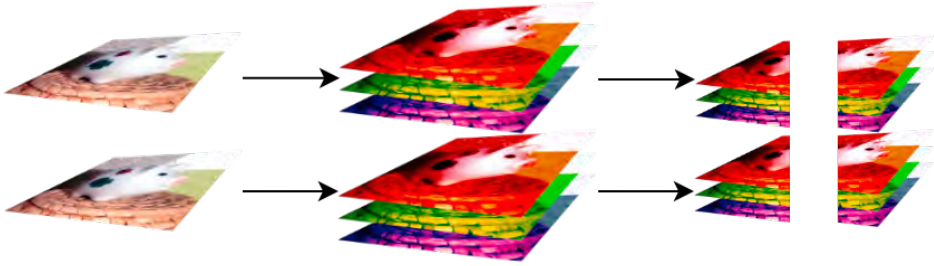

Fig. 7. Image preparation steps for benchmarking.

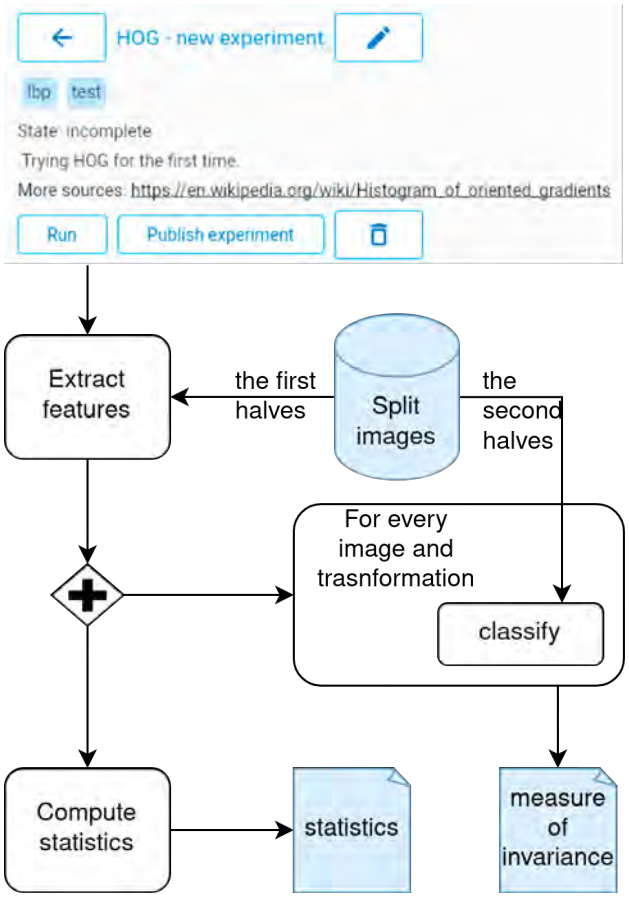

Fig. 8. The benchmark process diagram.

6. Conclusion

We have analyzed a variety of feature types with a focus on finding similarities in the feature structures in order to be able to save them in a generalized format. This effort led us to divide them into two main groups based on how the features represent data. The first one was *per-pixel* representation that was further specified as vectors for each pixel. The second group was *per-window*, which we divided more finely into four subgroups: vector, histogram, list, and big matrix. This solution created an abstraction layer in which it is not further important for benchmarking to know which specific feature is being used, but rather which of the groups and subgroups the feature data belongs. That leads to further simplification of benchmarking.

Furthermore, we have discussed how to measure feature properties and combine them with multispectral (or hyperspectral) images in a general way. We divided the properties into two categories, statistics and the extent of invariancy to image transformations (a measure of invariancy). With statistics, one can better understand the structure of the features, and with the measure of invariance, we can evaluate the robustness of features.

Finally, we combine the described methodology in a generalized benchmarking process for images with various spectra and a vast extent of features. This benchmarking process is being implemented in a web app. The web application is a portal where users can create experiments and evaluate implemented features after uploading them in a predefined format. Thanks to usability testing, the online application is user-friendly and a helpful tool for researchers to use.

References

[1] Aguilera, C., Barrera, F., Lumbreras, F., Sappa, A.D., Toledo, R., 2012. Multispectral image feature points. Sensors 12, 12661–12672. URL: https://www.mdpi.com/1424-8220/12/9/12661, doi:10.3390/s120912661.

[2] Ahonen, T., Matas, J., He, C., Pietikäinen, M., 2009. Rotation invariant image description with local binary pattern histogram fourier features, in: Scandinavian conference on image analysis, Springer. pp. 61–70.

[3] Bay, H., Ess, A., Tuytelaars, T., Van Gool, L., 2008. Speeded-up robust features (surf). Computer vision and image understanding 110, 346–359.

[4] Calonder, M., Lepetit, V., Strecha, C., Fua, P., 2010. Brief: Binary robust independent elementary features, in: Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11, Springer. pp. 778–792.

[5] Dalal, N., Triggs, B., 2005. Histograms of oriented gradients for human detection, in: 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05), Ieee. pp. 886–893.

[6] Daugman, J.G., 1993. High confidence visual recognition of persons by a test of statistical independence. IEEE transactions on pattern analysis and machine intelligence 15, 1148–1161.

[7] Fu, X., Wei, W., 2008. Centralized binary patterns embedded with image euclidean distance for facial expression recognition, in: 2008 Fourth International Conference on Natural Computation, IEEE. pp. 115–119.

[8] Guo, Z., Zhang, L., Zhang, D., 2010. A completed modeling of local binary pattern operator for texture classification. IEEE transactions on image processing 19, 1657–1663.

[9] Hafiane, A., Seetharaman, G., Zavidovique, B., 2007. Median binary pattern for textures classification, in: International Conference Image Analysis and Recognition, Springer. pp. 387–398.

[10] Haindl, M., 2022. Textural features.

[11] Haralick, R.M., Shanmugam, K., Dinstein, I.H., 1973. Textural features for image classification. IEEE Transactions on systems, man, and cybernetics , 610–621.

[12] Hauta-Kasari, M., Parkkinen, J., Jaaskelainen, T., Lenz, R., 1996. Generalized co-occurrence matrix for multispectral texture analysis, in: Proceedings of 13th International Conference on Pattern Recognition, IEEE. pp. 785–789.

[13] Kreutz, M., Völpel, B., Janßen, H., 1996. Scale-invariant image recognition based on higher-order autocorrelation features. Pattern Recognition 29, 19–26.

[14] Laws, K.I., 1980. Rapid texture identification, in: Image processing for missile guidance, SPIE. pp. 376–381.

[15] Liao, S., Law, M.W., Chung, A.C., 2009. Dominant local binary patterns for texture classification. IEEE transactions on image processing 18, 1107–1118.

[16] Lowe, D., 1999. Object recognition from local scale-invariant features, in: Proceedings of the Seventh IEEE International Conference on Computer Vision, pp. 1150–1157 vol.2. doi:10.1109/ICCV.1999.790410.

[17] Mikeš, S., Haindl, M., 2022. Texture segmentation benchmark. IEEE Transactions on Pattern Analysis and Machine Intelligence 44, 5647–5663. doi:10.1109/TPAMI.2021.3075916.

[18] Ojala, T., Pietikainen, M., Maenpaa, T., 2002. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Transactions on pattern analysis and machine intelligence 24, 971–987.

[19] Ojala, T., Pietikäinen, M., Harwood, D., 1996. A comparative study of texture measures with classification based on featured distributions. Pattern Recognition 29, 51–59. URL: https://www.sciencedirect.com/science/article/pii/0031320395000674, doi:https://doi.org/10.1016/0031-3203(95)00067-4.

[20] Remeš, V., Haindl, M., 2019. Bark recognition using novel rotationally invariant multispectral textural features. Pattern Recognition Letters 125, 612–617. URL: https://www.sciencedirect.com/science/article/pii/S0167865519301886, doi:https://doi.org/10.1016/j.patrec.2019.06.027.

[21] Rosten, E., Drummond, T., 2006. Machine learning for high-speed corner detection, in: Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9, Springer. pp. 430–443.

[22] Rublee, E., Rabaud, V., Konolige, K., Bradski, G., 2011. Orb: An efficient alternative to sift or surf, in: 2011 International conference on computer vision, Ieee. pp. 2564–2571.

[23] Tamura, H., Mori, S., Yamawaki, T., 1978. Textural features corresponding to visual perception. IEEE Transactions on Systems, man, and cybernetics 8, 460–473.

[24] Varma, M., Zisserman, A., 2005. A statistical approach to texture classification from single images. International journal of computer vision 62, 61–81.