OpenSea - Open Search Based Classification Tool

Konstantin Pogorelov
Simula Research Laboratory, Norway
University of Oslo, Norway

Zeno Albisser
Simula Research Laboratory, Norway
University of Oslo, Norway

Olga Ostroukhova
Research Institute of Multiprocessor Computation Systems n.a. A.V.Kalyaev, Russia

Mathias Lux
Klagenfurt University, Austria

Dag Johansen
UiT–The Arctic University of Norway

Pål Halvorsen
Simula Metropolitan Center for Digital Engineering, Norway
University of Oslo, Norway

Michael Riegler
Simula Metropolitan Center for Digital Engineering, Norway
University of Oslo, Norway

ABSTRACT

This paper presents an open-source classification tool for image and video frame classification. The classification takes a search-based approach and relies on global and local image features. It has been shown to work with images as well as videos, and is able to perform the classification of video frames in real-time so that the output can be used while the video is recorded, playing, or streamed. OpenSea has been proven to perform comparable to state-of-the-art methods such as deep learning, at the same time performing much faster in terms of processing speed, and can be therefore seen as an easy to get and hard to beat baseline. We present a detailed description of the software, its installation and use. As a use case, we demonstrate the classification of polyps in colonoscopy videos based on a publicly available dataset. We conduct leave-one-out-cross-validation to show the potential of the software in terms of classification time and accuracy.

CCS CONCEPTS

• **Information systems → Information retrieval**; • **Computing methodologies → Parallel algorithms**; **Search methodologies**; **Computer vision problems**; **Machine learning**;

KEYWORDS

Image, video, indexing, information retrieval, global features, machine learning, classification

ACM Reference format:
Konstantin Pogorelov, Zeno Albisser, Olga Ostroukhova, Mathias Lux, Dag Johansen, Pål Halvorsen, and Michael Riegler. 2018. OpenSea - Open Search Based Classification Tool. In *Proceedings of 9th ACM Multimedia Systems Conference, Amsterdam, Netherlands, June 12–15, 2018 (MMSys'18),* 6 pages.
https://doi.org/10.1145/3204949.3208128

Contact author's address: Konstantin Pogorelov, Simula Research Laboratory, Oslo, Norway, email: konstantin@simula.no .

MMSys'18, June 12–15, 2018, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5192-8/18/06…$0.00
https://doi.org/10.1145/3204949.3208128

1 INTRODUCTION

In the last years, multimedia data has become increasingly popular and important. More recently, big data is now a buzzword in the community related to the massive amount of multimedia data that becomes available because every user can create their own content and share it. However, also in other fields like medicine, the use of multimedia data, especially videos and images, has gained importance. This leads to a need for software and methods that make possible to search, categorize and classify this data efficiently based on content and not just based on metadata. Without such methods, the data available cannot be used efficiently. An example that leads to a lot of video data generation in the medical field is the use of camera pills (wireless video capsules), which traverse a patient's gastrointestinal (GI) tract. For a single patient, a camera pill collects between 4 and 12 hours of video material. Since medical experts are already overloaded, they do not have time to watch all the videos when the use of camera pills increases. Furthermore, batch processing of a huge amount of data costs resources and time, which are not always available. For cancer patients, it can be life-saving if their data is processed faster.

Therefore, we present an open-source system that provides a fast and easy way of classifying videos or images. It allows to easily create search-based classifiers that use global content features describing the image or frame as a whole. The OpenSea software contains a pipeline that allows the extraction of global features, creates indexes that are used as models for the classifier, classifies images and videos, and outputs the results in a format that can easily be used in a lot of various scenarios and applications by different users. Apart from that, OpenSea can also beat state-of-the-art methods such as convolutional neural networks (CNNs) in some use cases which makes it an easy to get and relatively hard to beat baseline for evaluation of approaches [6–8, 14]. It is also much faster compared to deep learning approaches [9]. Therefore, we believe that our tool is very useful for:

- Researchers who are not very familiar with classification, indexing or global features, but who want to use such methods.
- Researchers that develop state-of-the-art methods like deep learning architectures and want to benchmark their method with an easy to get and hard to beat baseline.
- Content managers and experts like medical doctors who want to use classification to learn from their data or understand their data better without going into technical details.

In addition, it can be useful for researchers who:

- Need a fast and reliable classification method that is easy to modify.
- Need just parts of the system like extraction of features, classification, etc.
- Want to evaluate their own classification methods based on comparison with our tool as an alternative to for example random baseline, etc.
- Work with big data and classification and segmentation problems.

In the remainder of the paper, we present the tool and how it can be used. We then present it in a medical use case to show its practical applicability.

2 THE SYSTEM

The tool that we have built consists of two separate parts, an *Indexer* and a *Classifier*, both parts are written in Java. To be able to extract image features from the content, we use the well known libraries OpenCV[1], Apache Lucene[2] and LIRE[3]. The *Indexer* can be used to create an index of images contained in a directory. The *Classifier* is able to read and process videos and images, and it uses feature similarities to perform a binary classification of frames in video or images in an index. The classification is done by identifying the most similar images in ground-truth indexes, which are provided as command line arguments.

2.1 Indexing and Training

The classifier uses indexes containing image descriptors of positive and negative examples as a model. Therefore, the classifier is trained by simply dividing negative and positive examples into the respective indexes. The Indexer accepts a list of directories as input in the command line. Each of the provided directories is then searched for image files, and an index of all the images contained in a directory is created. The index of all the images contained in a single directory is stored in a subdirectory called *index* in the form of Lucene-based indexes. The index can store multiple LIRE-feature-values per image, and the list of feature-values to store is provided in the command line. The supported features are all the features supported by LIRE [4] library. It is also built in a way that it can easily be extended in the case that one of the used libraries provides new features.

The usage of Lucene-based indexes has several advantages. These indexes are easy to compute and do not require a lot of storage space. Further, the indexes are optimized for search operations and can therefore be accessed efficiently. To increase the efficiency and the processing speed of our Indexer further, we have parallelized the indexing process. We create multiple threads that read image files from disk and calculate global image features concurrently. The results are then combined in a single index. The threads share the same list of files, but as the number of threads is fixed and known to each thread, we can split each video frame or image file statically. Every thread starts reading with an offset into one file and continues reading at offsets depending on its own thread ID. This allows us to implement reading without explicit locking of the input file list. Moreover, assuming that all images are of the same size, the workload is spread evenly across all threads. The actual number of threads used depends on the available processors reported by the Java Virtual Machine (JVM).

2.2 Classification of Video and Images

The classification of each video frame or image is based on the analysis of search results for a given query picture. The classification algorithm is a modified K-Nearest-Neighbor algorithm (k-NN). K-NN is a non-parametric algorithm, which means that the algorithm uses the rank of the values rather then the parameters of each frame. The frame classification is based on its k nearest neighbors by a majority decision. The classification algorithm used in the system differs in some points from the original k-NN algorithm. The first difference is that the algorithm is based on a ranked list of search results, which can be generated in real-time or pre-indexed for each query frame of the video. The second is that weighted values are used for generating a decision antithetical to the non-parametric behavior of the k-NN. The weights are based on the search result's ranked list. This part is designed in a way that it can easily be replaced with other different methods (for example visual page rank, etc.).

As mentioned before, the classification tool is implemented as a search for similar images in indexes that are generated off-line or on-the-fly, based on single or multiple image features. For every image in the input index or video, it searches the provided classifier indexes and finds the images with the most similar image features, whereas similarity is determined based on the low level features and their associated distance (in this case Tanimoto distance). Based on the class of the similar images retrieved from the index, the input image is classified. The result for every single image feature, as well as the result of late fusion for all the selected image features is displayed on-screen. Late fusion means that each feature has an own classification step that is combined with other classifiers' output for the final result. When classifying previously indexed images, an HTML page is created with a visual representation of all the classified images. When classifying a video sequence, the results are stored to a file in JSON format instead. The classification tool also determines the performance of the classification and calculates several evaluation scores such as *precision, recall, weighted f1-score, etc.*. For this to work, the input data must be labeled correctly before it is classified. This can either be done by prefixing the filenames of the files in the test index with 'p' or 'n' for positive and negative samples, respectively, or by supplying separate test indexes with the command line options for the input data.

[1] http://opencv.org/ [last visited, Feb. 10, 2018]
[2] http://lucene.apache.org/ [last visited, Feb. 10, 2018]
[3] http://www.lire-project.net/ [last visited, Feb. 10, 2018]

3 INSTALLATION AND LICENSE

OpenSea is licensed under the terms of the GNU General Public License (GPL) version 3, as published by the Free Software Foundation. OpenSea depends on LIRE, which is licensed under GPL version 2, and OpenCV, which is licensed under a BSD license.

We have tested our software on *Linux*, *Mac OS X* and *Windows*. For simplicity, we provide installation instructions for *Ubuntu Linux*. All the required files from stable LIRE version 0.9.5 and Apache Lucene distribution are already included into the OpenSea distribution. The following installation and build instructions were tested with *Ubuntu 16.04*:

- Download and install the Java SE Development Kit 8 from http://www.oracle.com.
- Make sure to have the directory containing the java compiler in your PATH environment variable.
- Install *OpenCV-Java* and *Apache ant*:
  ```
  sudo apt-get install libopencv2.4-java ant
  ```
- Clone the OpenSea repository:
  ```
  git clone \
      https://github.com/acmmmsys/2018-OpenSea
  ```
- Build OpenSea using *ant* as command line arguments.
  ```
  ant dist
  ```
  Once building finished, you should find the two files *classifier.jar* and *indexer.jar* in the subdirectory *dist*.
- To make sure the *OpenCV-Java* native libraries are found at runtime, it is further necessary to add the path to *libopencv_java249.so* to *LD_LIBRARY_PATH*.
  ```
  export LD_LIBRARY_PATH=/usr/lib/jni
  ```

If another versions of LIRE is required, the following additional steps are required:

- Download *Lire* from http://www.lire-project.net/.
- Unzip *Lire* to a directory of your choice. We will refer to this location as *Lire directory*.
- Make sure your *LIRE directory* contains the file *lire.jar*.
- Build OpenSea using *ant*, passing your *Lire directory* and the corresponding *OpenCV-Java directory* as command line arguments. The *OpenCV-Java directory* is where your java bindings for *OpenCV* were installed (used by both LIRE and OpenSea). It must contain the file *opencv-249.jar*, or any later version.
  ```
  ant -Dlire=/home/me/Lire \
      -Dopencv=/usr/share/OpenCV/java dist
  ```
  Once building finished, you should find the two files *classifier.jar* and *indexer.jar* in the subdirectory *dist*.

4 USAGE INSTRUCTIONS AND EXAMPLE

To show how to use OpenSea, we provide the usage instructions and a few command line examples.

4.1 Indexing

The indexer can be started as follows:

```
java \
        -jar [/path/to/jar/file/]indexer.jar
        [-f feature]
        /dir/with/images
        [/dir/with/more/images]
```

Indexer support multiple features set by *-f* command line argument as well as multiple directories with images or frames extracted from video.

Usage example:

```
java \
        -jar indexer.jar -f JCD -f FCTH \
        /home/user/dataset/train/pos \
        /home/user/dataset/train/neg \
        /home/user/dataset/test
```

This creates the two indexes containing the global image features *Joint Composite Descriptor* (JCD) and *Fuzzy Color and Texture Histogram* (FCTH) of the images in the */home/user/dataset/train/pos*, */home/user/dataset/train/neg /home/user/dataset/test* directories and stores the indexes in */home/user/dataset/train/pos/index*, */home/user/dataset/train/neg/index* and */home/user/dataset/test/index* directories respectively. If the index target directories contain any previously extracted features they will be replaced.

4.2 Classification

The classifier can be started as follows:

```
java \
        [-Djava.library.path=/path/to/opencv/for/
            java]
        -jar [/path/to/jar/file/]classifier.jar
        [-f feature]
        [-c /dir/with/training/index]
        [-p /dir/with/training/positive/index]
        [-n /dir/with/training/negative/index]
        [-i /dir/with/test/index]
        [-P /dir/with/test/positive/index]
        [-N /dir/with/test/negative/index]
        [-v /path/to/video/file]
```

Classifier support multiple features set by *-f* command line argument. Training and test datasets are expected to be supplied in the indexes previously extracted by Indexed. Indexes can be either joint or separated sets of positive and negative samples. For the joined sets file names must start with 'p' for positive sample and with 'n' for negative samples, with corresponding *-c* and *-i* command line arguments for training and test sets respectively. For the separated sets positive and negative samples must be provided in the separate indexes, with corresponding pairs *-p*, *-n* and *-P*, *-N* of command line arguments for training and test sets respectively. Classifying of video frames is implemented via *-v* command line arguments which is mutually exclusive with test set arguments.

Usage example:

```
java \
        -Djava.library.path=/usr/lib/jni \
        -jar classifier.jar \
        -f JCD -f FCTH -f Tamura \
    -p /home/user/dataset/train/pos/index \
    -n /home/user/dataset/train/neg/index \
    -i /home/user/dataset/test/index
```

This example shows how to classify images from the index */home/user/dataset/test/index* using the image features *JCD* and *FCTH*, by finding the most similar images among the positive samples from */home/user/dataset/train/pos/index* and the negative samples from

/home/user/dataset/train/neg/index. For the calculation of the evaluation metrics, it is required that the images indexed in */home/user/-dataset/test/index* have names starting with 'p' or 'n' for positive or negative samples, respectively. This generates visual classification output in HTML format. Example of generated HTML is depicted in figure 1.

```
java \
        -Djava.library.path=/usr/lib/jni \
        -jar classifier.jar \
        -f JCD \
    -p /home/user/dataset/train/pos/index \
    -n /home/user/dataset/train/neg/index \
    -P /home/user/dataset/test/pos/index \
    -N /home/user/dataset/test/neg/index \
    -f JCD
```

The second example uses samples from the positive index */home-/user/dataset/test/pos/index* and negative samples from the negative index */home/user/dataset/test/neg/index*, which are classified using the image feature *JCD*. The previously known classification is only used for evaluating the results of the classifier.

```
java \
        -Djava.library.path=/usr/lib/jni \
        -jar classifier.jar \
        -f JCD \
    -p /home/user/dataset/train/pos/index \
    -n /home/user/dataset/train/neg/index \
    -v /home/user/dataset/testvideo.avi
```

In our last example, a video file is supplied as input to the classifier. All video frames of this input video */home/user/dataset/testvideo.avi* are classified by searching the most similar images among the positive samples from */home/user/dataset/train/pos/index* and the negative samples from */home/user/dataset/train/neg/index* using the global image feature *JCD*. In addition to the on-screen output (see figure 2 for an example), a JSON file is generated, which contains a list of all the positive frames and a list of all the negative ones.

To process videos in real-time, we have also parallelized the classifier. Again, the number of threads created depends on the number of processors reported by the JVM. Each thread holds a separate instance of the classifier indexes, but all threads share the same queue for the input data to be classified. Therefore, every image or video frame is only loaded once, it is then processed by a single thread, and the result is written to a shared data structure. This allows for all threads to operate independently, with only two critical sections, i.e., one for dequeuing the next input image and one for writing to the shared result data structure. When processing a video as input data, an additional thread is created for reading the video from a file and filling the input frame queue. The Classifier tool further provides different options for weighting the count or distance score of similarity results. The different weighting methods can be chosen by adding the flag $-m$ followed by the rank method that should be applied to the command. As default mode, no weight is set and the classifier uses only the count per class. At the moment, we support 3 additional weighting methods: (i) weighted by rank position, i.e., the weight is computing from the position in the returned ranked list, (ii) weighted by distance, which uses the Tanimoto distance from the search as weight and (iii) weighted by average distance, which uses the average distance of all returned

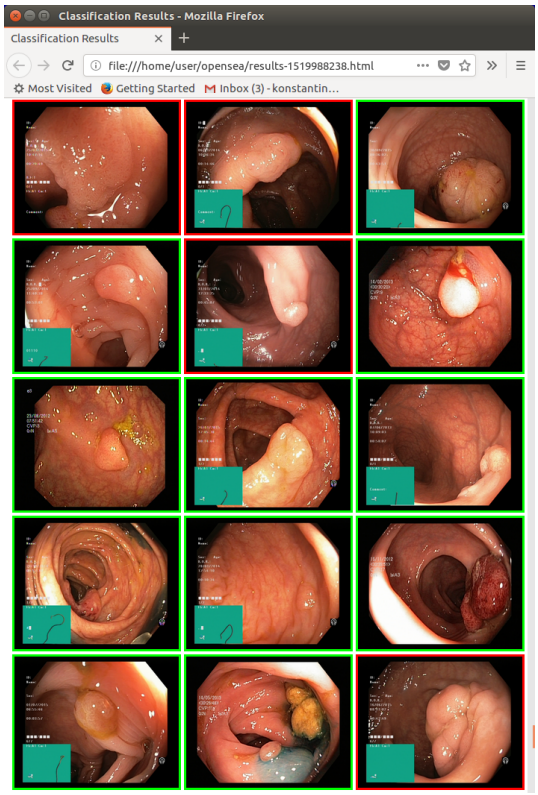

Figure 1: Example of a generates visual classification output in HTML format. Images with green borders correspond to true positive and true negative samples. Images with red borders correspond to false positive and false negative samples.

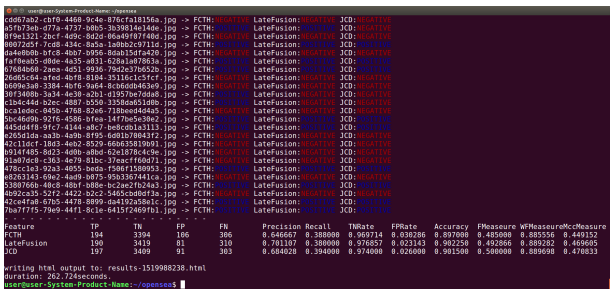

Figure 2: Example of a classifier on-screen output. The output contains classification results for each used feature and features' late fusion, as well as the corresponding performance metrics.

documents in the ranked list instead of the number of documents to calculate the weight. Moreover, various different combinations of global image features can be evaluated separately or combined in late fusion. This makes the tool ideal for experimenting with different approaches and finding an optimal set of features to use for a specific use case.

4.3 Metrics

In our performance evaluation, several specific metrics are implemented and can be calculated for the test data T. All metrics are calculated based on true positives (tp), false positives (fp), true negatives (tn), false negatives (fn) values per class $c \in T$. The most important metrics are precision, recall, $F1$ score and Weighted $F1$ score. A common and often used metric to calculate the quality of a classifier that considers both measures, *precision* and *recall*, is the $F1$ score. It is the harmonic mean (mean value of a number of values) of *precision* and *recall*. One problem with the standard $F1$ score is that a low value is not always an indicator of a badly performing classifier or retrieval system if the classes in the test dataset are not normally distributed [13]. To solve this problem, the weighted $F1$ score (WF1) can be used. This score takes both, the negative and the positive class results, into account and calculates a more accurate and robust measure. $WF1$ score is known to be more reliable to evaluate the performance of a classifier or retrieval system than the standard $F1$ score. Apart from $F1$ and $WF1$, the tool also provides true negative rate, false positive rate, accuracy and the Matthews correlation coefficient [5].

All of these metrics are suitable for showing the performance of binary (two classes) and multi-class classifiers (more than two classes) and should be a valuable set of instruments for users of the tool to evaluate their classifiers.

5 USE CASE

To show how the system works, we performed two experiments using two different pairs of training and test sets. For the first experiment, we used the ASU-Mayo Clinic polyp database [2]. It is at the moment one of the largest publicly available dataset of colonoscopy videos. The dataset comes with a ground truth that indicates if a frame of a video contains a polyp in the colon or not. The dataset consist of 20 videos. 10 videos do not contain polyps at all, and 10 of them contain polyps in the whole video or parts of it.

First, we split the dataset into test and training sets. The test set contained two separate videos that are not used in the training dataset. To measure the performance, we used the well known metrics precision and recall. All the tests were conducted without a weighting method (default mode). In this first test, we achieved a precision of 0.903, a recall of 0.919. For these results, we used a fusion of the features JCD and OpponentHistogram, which we found to perform best in small tests before [15]. The number of visual neighbors (size of the rank list returned by the search part of the classifier) was 71. The majority class baseline (all negative) is $0,683$ for precision, $0,683$ for recall.

To evaluate the robustness of the classifier, and to check if the good results were not just overfitting, we decided to perform a leave-one-out-cross-validation (LOOC) with all 20 videos of the dataset. In LOOC, all videos of the dataset are used to train the model expect for one that is used as the test example. This is repeated, so that all the sample videos are excluded once. To be able to recreate the experiments and test the software, we added the indexes to the official repository. We used the same features and number of visual neighbours as in the test before. For LOOC, the average precision is $0,895$, the average recall is 0.903. In comparison to the LOOC for the majority class baseline (all negative) which reaches a precision of 0.636, recall of 0.636. It is important to point out that we choose the class with the highest number for the majority vote baseline against the common practice to decide for the positive one. This makes it harder to outperform the baseline, but it also shows the real performance of the classifier. The results shows that our system performs well in cross validation and that it is robust and not overfitted for the dataset. We also want to point out that the classification time is very low. For a single frame, the time is around 30 milliseconds. To be able to do it in real time for videos with 30 frames per second, $33,3$ milliseconds is the deadline. In the best case, if we use a single feature, we can even get a classification time of around 10 milliseconds. The parallelization is not yet optimized, and we have some ideas that can make the system even faster, but this is out of scope for this paper.

Table 1: A performance comparison of deep-learning and global features based GI findings detection approaches [9]

	Global-features-based EIR	*Deep-EIR*
Detection Type	polyps / 30 features	abnormalities / neural network
Recall (Sensitivity)	98.50%	87.20%
Precision	93.88%	87.20%
Specificity	72.49%	97.40%
Accuracy	87.70%	97.50%
FPS	**300**	**30**

For the second experiment, we use combinations of four different, publicly available datasets, namely CVC-356 [2], CVC-612 [1], Kvasir [8] and parts of Nerthus [7]. The CVC-356 and CVC-612 datasets consist of 356 and 612 video frames, respectively. Each frame that contains a polyp comes with pixel-wise annotations in the CVC-356 and CVC-612 datasets. They both are used for training only in our polyp detection experiments. The fames from those datasets were renamed adding 'n' or 'p' prefix to reflect actual polyp presence in frames according to the existing pixel-wise annotations. For the testing we used Kvasir and Nerthus. For the Kvasir dataset, we included all classes except for the dyed classes (in a real world scenario something dyed is already detected by the doctor) leading to a dataset containing 1,000 frames with polyps, 5,000 without. We also added the 1,350 of class three frames with normal mucosa from the Nerthus dataset.

For this experiment we performed training and polyp detection using the described sets with two different detection approaches: the proposed OpenSea system and a deep-learning based abnormality detection approach [9]. The comparison of performance and data processing speed is depicted in table 1. As one can see, the OpenSea (global-features-based EIR) approach can perform as good in terms of detection performance as deep-learning based, but OpenSea system perform ten times faster in terms of processing speed. This results showing a promising nature of global features and their ability to perform fast and efficiently even across the different datasets.

The problem of polyp detection in GI videos is one of the most important problems in modern medical endoscopic imaging analysis [6]. Our efforts in this field includes not only development of the new lesion recognition methods [9], but also include a creation of open and publicly available datasets. We are working intensively on extending our own datasets which contain another diseases and findings [7, 8]. The proposed OpenSea system can easily be extend to different diseases by simply using a separate classifier for each

category which will make it easy to run in parallel and more accurate (since it is late fusion and late fusion has been proved as being more accurate [3]). The preliminary results of such a multi-class classification can be found in [10].

It is important to point out that with our method the adjustment of precision and recall is very easy. We can easily increase the recall by using more visual neighbors. This makes it very interesting for the medical use case, because we can get a recall of 1 so that doctors can be sure that we do not miss a true positive example, while still saving them working time because the high precision allows to remove a considerable number of fames.

Possible ways to use the output of the classification tool are presented in the following papers. Here, we use it in a system that allows computer-aided diagnosis. It helps medical experts to find polyps in colonoscopies and also to save medical personnel's working time because they do not have to analyze the whole video. OpenSea has been also used for a system called EIR. This system is built to automatically detect different disease during colonoscopies and capsular endoscopies. The more detailed description of the system can be found in [9, 11, 15]. Different demos of this system have been presented in [12, 16].

Comparing to another existing classification-related software (e.g. Weka[4], a collection of machine learning algorithms for data mining tasks), OpenSea provides not only classification capabilities, but integrates them with feature extraction process. This integration and the simplified data annotation mechanism make the OpenSea tool easy-to-use for all user categories including non-expert users and professionals.

6 CONCLUSION

We presented an easy to use open-source software named OpenSea for image and video classification and showed that the performance regarding processing time and detection accuracy is promising. By making the tool open-source, we hope that we can help other researchers to compare their systems and develop better methods by being able to use it as an easy to get but hard to beat baseline. Moreover, due to the easy way to train the classifier, we hope that also non-experts can use it, especially in the medical use case that we presented. For the future project development, we plan to integrate OpenSea with the latest version of LIRE, speed-up the features extraction process [11], add more features and metrics, integrate custom weights and extend the report generation capabilities.

REFERENCES

[1] Jorge Bernal, F Javier Sánchez, Gloria Fernández-Esparrach, Debora Gil, Cristina Rodríguez, and Fernando Vilariño. 2015. WM-DOVA maps for accurate polyp highlighting in colonoscopy: Validation vs. saliency maps from physicians. *Computerized Medical Imaging and Graphics* 43 (2015), 99–111.

[2] Jorge Bernal, Nima Tajbakhsh, Javier Sanchez, Bogdan J Matuszewski, Hao Chen, Lequan Yu, Quentin Angermann, Olivier Romain, Bjorn Rustad, Ilangko Balasingham, Konstantin Pogorelov, Sungbin Choi, Quentin Debard, Lena Maier-Hein, Stefanie Speidel, Danail Stoyanov, Patrick Brandao, Henry Córdova, Cristina Sánchez-Montes, Suryakanth R. Gurudu, Gloria Fernández-Esparrach, Xavier Dray, Jianming Liang, and Aymeric Histace. 2017. Comparative Validation of Polyp Detection Methods in Video Colonoscopy: Results from the MICCAI 2015 Endoscopic Vision Challenge. *IEEE Transactions on Medical Imaging* (2017), 1–19.

[3] Hugo Jair Escalante, Carlos A Hérnadez, Luis Enrique Sucar, and Manuel Montes. 2008. Late fusion of heterogeneous methods for multimedia image retrieval. In *Proc. of ACM ICMR*. 172–179.

[4] M. Lux and O. Marques. 2013. *Visual Information Retrieval Using Java and LIRE*. Vol. 25. Morgan & Claypool.

[5] Brian W Matthews. 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405, 2 (1975), 442–451.

[6] Konstantin Pogorelov, Sigrun Losadal Eskeland, Thomas de Lange, Carsten Griwodz, Kristin Ranheim Randel, Håkon Kvale Stensland, Duc-Tien Dang-Nguyen, Concetto Spampinato, Dag Johansen, Michael Riegler, and Pål Halvorsen. 2017. In *Proc. of MMSys*. 112–123.

[7] Konstantin Pogorelov, Kristin Ranheim Randel, Thomas de Lange, Sigrun Losada Eskeland, Carsten Griwodz, Dag Johansen, Concetto Spampinato, Mario Taschwer, Mathias Lux, Peter Thelin Schmidt, Michael Riegler, and Pål Halvorsen. 2017. Nerthus: A Bowel Preparation Quality Video Dataset. In *Proc. of MMSYS*. 170–174.

[8] Konstantin Pogorelov, Kristin Ranheim Randel, Carsten Griwodz, Sigrun Losada Eskeland, Thomas de Lange, Dag Johansen, Concetto Spampinato, Duc-Tien Dang-Nguyen, Mathias Lux, Peter Thelin Schmidt, Michael Riegler, and Pål Halvorsen. 2017. Kvasir: A Multi-Class Image Dataset for Computer Aided Gastrointestinal Disease Detection. In *Proc. of MMSys*. 164–169.

[9] Konstantin Pogorelov, Michael Riegler, Sigrun Losada Eskeland, Thomas de Lange, Dag Johansen, Carsten Griwodz, Peter Thelin Schmidt, and Pål Halvorsen. 2017. Efficient disease detection in gastrointestinal videos - global features versus neural networks. *Multimedia Tools and Applications* 76, 21 (2017), 22493–22525.

[10] Konstantin Pogorelov, Michael Riegler, Pål Halvorsen, Carsten Griwodz, Thomas de Lange, Kristin Randel, Sigrun Eskeland, Dang Nguyen, Duc Tien, Olga Ostroukhova, and others. 2017. A comparison of deep learning with global features for gastrointestinal disease detection. In *Proc. of CEUR Workshop*.

[11] Konstantin Pogorelov, Michael Riegler, Pål Halvorsen, Peter Thelin Schmidt, Carsten Griwodz, Dag Johansen, Sigrun L. Eskeland, and Thomas de Lange. 2016. GPU-accelerated Real-time Gastrointestinal Diseases Detection. In *Proc. of CBMS*. 185–190.

[12] Konstantin Pogorelov, Michael Riegler, Jonas Markussen, Mathias Lux, Håkon Kvale Stensland, Thomas Lange, Carsten Griwodz, Pål Halvorsen, Dag Johansen, Peter T Schmidt, and Sigrun L. Eskeland. 2016. Efficient Processing of Videos in a Multi Auditory Environment Using Device Lending of GPUs. In *Proc. of MMSys*. 36.

[13] DMW Powers. 2011. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies* 2, 1 (2011), 37–63.

[14] Michael Riegler, Mathias Lux, Carsten Griwodz, Concetto Spampinato, Thomas de Lange, Sigrun L Eskeland, Konstantin Pogorelov, Wallapak Tavanapong, Peter T Schmidt, Cathal Gurrin, and others. 2016. Multimedia and Medicine: Teammates for Better Disease Detection and Survival. In *Proc. of ACM MM*. 968–977.

[15] Michael Riegler, Konstantin Pogorelov, Pål Halvorsen, Thomas de Lange, Carsten Griwodz, Peter Thelin Schmidt, Sigrun L. Eskeland, and Dag Johansen. 2016. EIR - Efficient Computer Aided Diagnosis Framework for Gastrointestinal Endoscopies. In *Proc. of CBMI*. 1–6.

[16] Michael Riegler, Konstantin Pogorelov, Jonas Markussen, Mathias Lux, Håkon Kvale Stensland, Thomas de Lange, Carsten Griwodz, Pål Halvorsen, Dag Johansen, Peter T Schmidt, and Sigrun L. Eskeland. 2016. Computer Aided Disease Detection System for Gastrointestinal Examinations. In *Proc. of MMSys*. 29.

[4] https://www.cs.waikato.ac.nz/ml/weka/ [last visited, Feb. 10, 2018]