Image classification using topological features automatically extracted from graph representation of images

Liping Yang
Diane Oyen
liping.yang@lanl.gov
doyen@lanl.gov
Computer, Computational, and Statistical Sciences Division, Los Alamos National Laboratory
Los Alamos, New Mexico

Brendt Wohlberg
brendt@lanl.gov
Theoretical Division, Los Alamos National Laboratory
Los Alamos, New Mexico

ABSTRACT

The performance of machine learning methods is strongly dependent on the data representation (features) to which they are applied. For drawings in particular, we cannot rely on texture, color or shading information; there is little information present in a drawing beyond the spatial relationships and topology. A topological graph is an intuitive and powerful data structure and data representation framework that can capture the topological relations and spatial arrangement among entities present in images. In this paper, we use topological features automatically extracted from graph representations of images for image classification. Our approach is simple, intuitive, and generic. We compare our method against a traditional feature descriptor, histogram of oriented gradients (HOG) on the MNIST data set. The results demonstrate the effectiveness of our graph approach, especially when applied to small sets of training data. In addition, our method is very fast to train, and also much less sensitive to hyperparameters, requiring little hyperparameter fine tuning.

CCS CONCEPTS

• **Computing methodologies** → **Computer vision**; *Machine learning*; Image processing; • **Mathematics of computing** → Graph theory.

KEYWORDS

graph theory, topological analysis, computer vision, machine learning, image analysis, image classification

ACM Reference Format:
Liping Yang, Diane Oyen, and Brendt Wohlberg. 2019. Image classification using topological features automatically extracted from graph representation of images. In *Proceedings of 15th International Workshop on Mining and Learning with Graphs (Anchorage '19).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Anchorage '19, Aug 5, 2019, Anchorage, Alaska

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION AND RELATED WORK

The performance of machine learning (ML) algorithms generally depends on the choice of data representation (or features) to which they are applied [2, 10]. In ML, feature learning or representation learning [2] is a set of techniques that allows a system to automatically discover the representations needed for feature detection or classification from raw data, but feature learning can require a very large set of training data. Feature engineering is a way to take advantage of human ingenuity and prior knowledge to compensate for that weakness. Thus, much of the actual effort in deploying ML algorithms goes into the design of pre-processing pipelines and data transformations that result in a representation of the data that can support effective ML [2, 10]. This feature engineering process is important but labor-intensive, and highlights the weakness of ML algorithms [31].

Representation learning approaches for ML on graphs offer a powerful alternative to traditional feature engineering. In recent years, for graph-structured data, representation learning on graphs has consistently improved performance on tasks such as node classification and link prediction [10]. We extend the use of graph representation learning to the image domain where the graph is constructed from data to represent spatial relationships and topology in the image. Graph representation of an image is a compact representation of the pertinent features of drawings and hand-written figures (as opposed to a pixel matrix); and therefore promises to reduce the sample complexity of classifying images.

Deep learning has achieved impressive accuracy for labeling and segmenting natural images, but this performance has not yet been realized for scientific and technical images. Deep learning architectures for image analysis use convolutional neural network (CNN) architectures which learn parameters that are translationally invariant and build up local features to broader scales. However, CNNs cannot represent spatial relations among objects contained in images, as shown in Figure 1. The capsule network, also called CapsNet [23] can capture hierarchical relationships in an image, but not more general spatial relationships, such as angles. A graph neural network (GNN) can capture relationships among objects, but the graph structure is fixed [10].

Recent work shows that topology and shape analysis are promising approaches for improving image classification exploiting local features [3, 6, 13, 17, 24]. Yet, for drawings and scene understanding, we expect relationships at the local and global scales to be important. To the best of our knowledge, derived features from

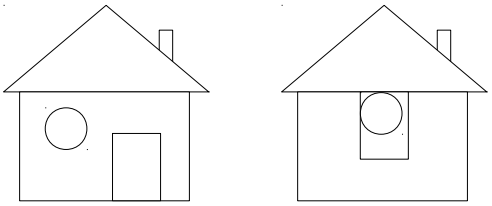

Figure 1: Why spatial relationships and arrangement among objects matter. CNN cannot tell the difference between the left and the right, because CNN does not retain spatial relations. A graph-based approach that preserves spatial relationships has clear and intuitive advantages to deal with such problems.

graph representations (such as number of leaf nodes, bridge number of graphs, etc.) are not used for features to train ML models – rather, the features are still in raster (pixel matrix) format in existing graph-based ML and deep learning methods.

Rather than learning features through deep learning, commonly-used invariant features for image processing and computer vision are local shape descriptors such as SIFT [18], SURF [1], and DAISY [25]. Yet, to efficiently compare and analyze shapes of objects in images, we need a data structure that can capture and store global shape information and connectivity. A graph provides the ideal solution [10].

Skeletonization, also called thinning, is an important preprocessing step in computer vision and image processing tasks such as shape analysis and vectorization. Thinning generates a compact representation of images called a skeleton. A skeleton is a central line extraction of an object via thinning [15]. The skeleton can be useful for feature extraction, and representation of objects' topology, because a skeleton captures essential topology and shape information of an object in a simple form [14]. The thinning process reduces redundant information in images and thus reduces image complexity for tasks such as shape analysis and scene understanding. Thus, in this work, we generate a skeleton graph from skeletons of images.

2 APPROACH

The core innovation of our approach is to train ML algorithms using automatically extracted topological features for image classification based on analyzing the topological relations stored in skeleton graphs of images. Figure 2 provides the workflow of our method.

Given an input image, the skeleton is extracted after preprecessing such as denoising. A skeleton graph (see Figure 3 for an illustrative example) is then generated from the image skeleton. After converting the image to a graph representation, the graph is analysed, and topological features such as cycles and bridges, are computed. In this work, topological features extracted from graphs and used for ML models includes the following 15 features: *diameter*, *radius*, *spectral radius*, *density*, *node connectivity*, *edge connectivity*, *transitivity*, *algebraic connectivity*, and the number of *leaf nodes*, *junction nodes*, *cycles*, *bridges*, *central points*, *nodes in periphery*, and *connected components* (some examples are listed in Table 1, and a complete list of definitions pertinent to the topological features extracted from graphs is provided in Appendix B).

The next step is to split the annotated data set into two subsets, one for training and the other for testing, with a splitting ratio of 70% to 30%. In general, there is not a universally better ML algorithm for all problems [7]. Thus, we train the images in the training set (with the extracted topological features) on three well-known ML algorithms, specifically, random forest (RF), support vector machine (SVM), and neural net (NN), for image classification.

To make each ML model perform their "best", before training the ML algorithms, we tune hyperparameters of each ML algorithm on the training set. Hyperparameter tuning is the process of finding the set of hyperparameter values of a ML algorithm that produces the best model results. In general, hyperparameter tuning is crucial because it is used to search for the best hyperparameters of a ML algorithm for a given dataset. We use k-fold cross-validation [12, 29] while fine-tuning the hyperparameters to avoid overfitting, because it allows performance variation across training sets to be examined and thus can test how well the trained ML algorithm is able to handle larger unseen population of data samples.

After fine-tuning, we use the best hyperparameters of each ML algorithm to train on the training set and the models are evaluated on the test set.

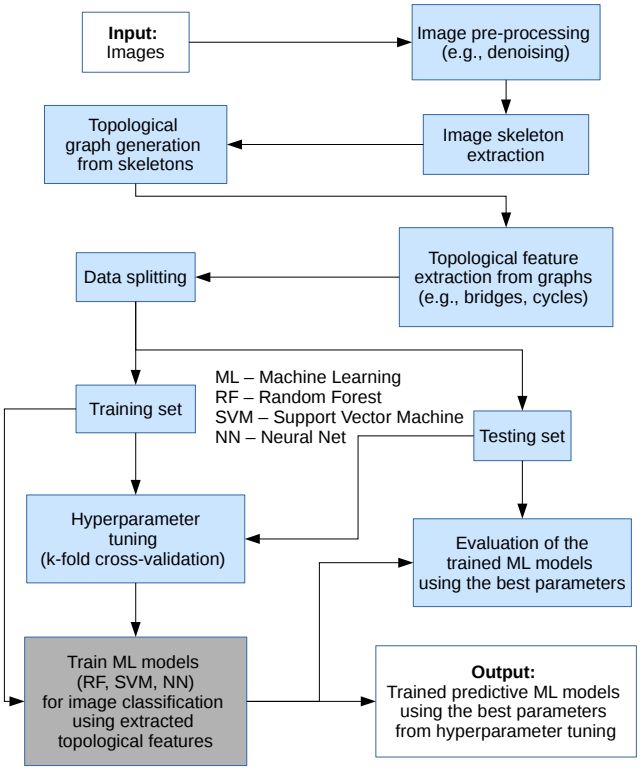

Figure 2: The workflow of our image classification approach using topological features.

3 EXPERIMENTS AND RESULTS

To test our method, we train on the MNIST data set for handwritten digits classification [16]. Note that the MNIST images are small and

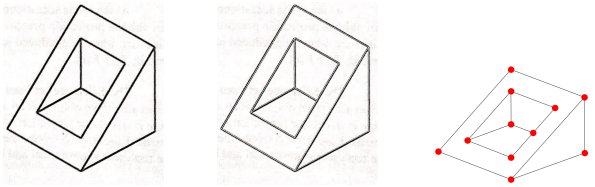

Figure 3: An illustrative example of skeleton and skeleton graph (left: input image, middle: skeleton drawn on the original image, right: skeleton graph generated from skeleton).

contain only a single object and so any number of ML approaches work well on this data. Yet, if the graph-based approach can correctly identify digits in this data, then we can expect to be able to identify digits within larger images through sub-graph matching.

In the experiments, we used the first 100, 500, 1000, 1500, and 2000 (training and test set splitting ratio 70% vs. 30%) handwritten digits examples from the training set of the MNIST dataset to train RF, SVM and NN models.

For the implementation, we used OpenCV, SymPy [19], Scikit-image [26], NetworkX [9], and Scikit-learn [20]. Since the extracted topological features are not image-like (i.e., a pixel matrix), the training process is much faster. Figure 4 shows examples of skeletons and skeleton graphs (drawn on the corresponding skeleton) representing digits from the MNIST dataset.

Tables 2, 3, and 4 provide the training results and comparison against traditional image-based features, histogram of oriented gradients (HOG), including how many percentage points of accuracy each algorithm improved compared with its corresponding baseline model that uses the default hyperparameters in Scikit-Learn (version 0.20.3). For the *training accuracy* and *testing accuracy* columns in the tables, the left value refers to accuracy score of the best model and right refers to that of the baseline model. We can see that almost all ML models have better performance when using the best hyperparameters from tuning, and some models are improved substantially (e.g., SVM shown in Table 3). From the tables, we can also see that RF classifier is best with our method, in terms of accuracy, training time and sensitivity to model hyperparameter.

From Tables 2, 3, and 4, the clear advantages of our method is that it can provide very good results with a small training data set, with very fast training time and very robust for hyperparameter tuning (i.e., its performance is much less dependent on hyperparameter tuning; see Table 3 for a clear example, the values of the last two columns of HOG features are much larger than those using topological features). Fine tuning is recommended in general, because the ML performance will be (much) better when using fine tuning. Yet, when using our graph approach, fine tuning does not improve the result much – which is good – as our approach reduces engineer and analyst burden in tuning hyperparameters for practical ML-based applications.

We used the Grid Search in Scikit-learn [20] to tune hyperparameters of the ML algorithms. Detailed settings about the hyperparameter tuning for each ML algorithm are provided in Appendix D, particularly Table 5; and we give tuning results including best hyperparameters and tuning time in Table 6. While comparing the model improvement using the best hyperparameters from tuning against the baseline model, we use 10-fold cross-validation. The main reason we set k as 10-fold is based on the conclusion in literature, as Witten et al. [29] introduced, tests on different data sets, with different learning techniques, have shown that 10 is about the right number of folds to get the best estimate of error.

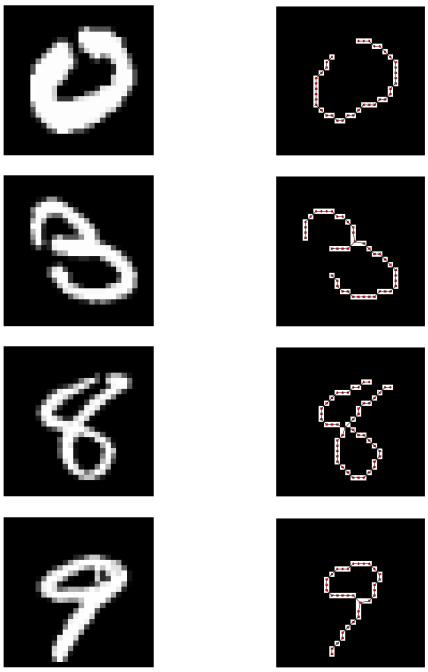

Figure 4: Examples of skeletons and skeleton graphs for MNIST data. The first column shows the image example, and the second shows the skeleton graph overlaid on the corresponding skeleton.

4 CONCLUSION AND FUTURE WORK

We propose an approach that uses global topological features extracted from the skeleton graph representation of images for image classification. Our approach is simple and generic. The approach is applied to the MNIST data set, and compared against the traditional feature descriptor, histogram of oriented gradients (HOG), and the experiment results show the effectiveness of our topological feature-based method for image classification. Specifically, our approach achieves very good and reliable results on small training data sets, and with very fast training and hyperparameter tuning processes, plus the performance of our approach is much less dependent on hyperparameter tuning.

In the future, we will extract more invariant topological and geometric features that can be extracted from graphs; such as geometry, including angles and orientations, to further improve ML performance. To further improve the performance (both speed and accuracy) of our method, we will use the Douglas-Peucker algorithm (also known as Ramer-Douglas-Peucker algorithm or iterative end-point fit algorithm) [8, 22] and Visvalingam's algorithm [27] to simplify the geometry in skeleton graphs (e.g., remove redundant points on a straight or near straight line segment) – this will make

Table 1: A few extracted topological features corresponding to the four examples shown in Figure 4. (# refers to *number of*. A *leaf node* is a node with its degree as 1, also called *end node* or *terminal node*. A *junction node* is a node whose degree >2. A *bridge* in an undirected connected graph is an edge that disconnects the graph if removing it. The *diameter* of a graph is the length of longest path of the shortest paths between any two nodes.)

example	# leaf nodes	# junction nodes	# cycles	# bridges	# connected components	diameter
row 1	2	0	0	36	1	36
row 2	3	1	0	41	1	37
row 3	2	1	1	24	1	24
row 4	1	1	1	9	1	32

Table 2: Results and comparison of ML algorithm performance. (RF: *Random Forest*; HOG: *Histogram of Oriented Gradients*; TF: *Topological Features*.)

ML Model	# Samples (train, test)	Training accuracy (best, baseline) (%)	Testing accuracy (best, baseline) (%)	Training time (best, baseline)	Improvement of training accuracy (%)	Improvement of testing accuracy (%)
RF (HOG)	100 (70,30)	70.00 (+/- 20.65), 45.71 (+/- 20.00)	60.00, 63.33	4 m 11 s, < 1 s	53.12	-5.26
	500 (350,150)	89.71 (+/- 2.91), 78.57 (+/- 8.40)	78.00, 69.33	41 m 39 s, < 1 s	14.18	12.50
	1000 (700,300)	92.00 (+/- 2.65), 82.00 (+/- 4.34)	89.33, 79.33	25 m 20 s, < 1 s	12.20	12.61
	1500 (1050,450)	92.19 (+/- 2.20), 86.00 (+/- 2.52)	92.44, 80.89	29 m 36 s, 1 s	7.20	14.29
	2000 (1400,600)	93.07 (+/- 3.77), 87.00 (+/- 3.03)	91.83, 85.50	20 m 17 s, 1 s	6.98	7.41
RF (TF)	100 (70,30)	88.57 (+/- 8.57), 84.29 (+/- 13.48)	90.00, 93.33	48 s, < 1 s	5.08	-3.57
	500 (350,150)	97.71 (+/- 2.14), 94.00 (+/- 2.98)	92.67, 92.00	55 s, < 1 s	3.95	0.72
	1000 (700,300)	96.00 (+/- 2.10), 94.57 (+/- 1.78)	97.33, 93.33	1 m 3 s, < 1 s	1.51	4.29
	1500 (1050,450)	95.33 (+/- 2.39), 95.24 (+/- 2.21)	96.67, 94.22	1 m 11 s, < 1 s	0.10	2.59
	2000 (1400,600)	95.86 (+/- 1.42), 94.79 (+/- 1.57)	96.33, 95.00	1 m 21 s, < 1 s	1.13	1.40

the graph analysis part more efficient). While Douglas-Peucker is the most well-known for simplifying geometry, Visvalingam's algorithm [27] is more effective and has a remarkably intuitive explanation: it progressively removes points with the least-perceptible change. We will investigate both algorithms and evaluate which one will work best for simplifying skeleton graphs.

Topological skeleton graph representation of images has clear and intuitive advantages for further advanced image analysis. Thus, many potential applications, such as intelligent image interpretation and understanding for visually impaired people in complex spaces, could benefit from the graph approach.

REFERENCES

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. 2008. Speeded-up robust features (SURF). *Computer vision and image understanding* 110, 3 (2008), 346–359.

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[3] Naeem Bhatti, Allan Hanbury, and Julian Stottinger. 2018. Contextual local primitives for binary patent image retrieval. *Multimedia Tools and Applications* (2018), 1–41.

[4] Gray Chartrand and Ortrud R Oellermann. 1993. Applied and algorithmic graph theory. (1993).

[5] Thomas F Coleman and Jorge J Moré. 1983. Estimation of sparse Jacobian matrices and graph coloring blems. *SIAM journal on Numerical Analysis* 20, 1 (1983), 187–209.

[6] Tamal Dey, Sayan Mandal, and William Varcho. 2017. Improved image classification using topological persistence. In *Proceedings of the conference on Vision, Modeling and Visualization*. Eurographics Association, 161–168.

[7] Pedro Domingos. 2015. *The master algorithm: How the quest for the ultimate learning machine will remake our world.* Basic Books.

[8] David H Douglas and Thomas K Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization* 10, 2 (1973), 112–122.

[9] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX.* Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[10] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

Table 3: Results and comparison of ML algorithm performance. (SVM: *Support Vector Machine*; HOG: *Histogram of Oriented Gradients*; TF: *Topological Features*.)

ML Model	# Samples (train, test)	Training accuracy (best, baseline) (%)	Testing accuracy (best, baseline) (%)	Training time (best, baseline)	Improvement of training accuracy (%)	Improvement of testing accuracy (%)
SVM (HOG)	100 (70,30)	78.57 (+/- 11.52), 11.43 (+/- 5.71)	70.00, 6.67	9 s, < 1 s	587.50	950.00
	500 (350,150)	92.86 (+/- 3.67), 9.71 (+/- 4.81)	82.67, 14.00	3 m 5 s, 11 s	855.88	490.48
	1000 (700,300)	94.00 (+/- 2.62), 28.43 (+/- 3.16)	91.67, 8.33	12 m 4 s, 42 s	230.65	1000.00
	1500 (1050,450)	95.24 (+/- 1.54), 49.33 (+/- 4.44)	92.00, 13.56	25 m 41 s, 1 m 36 s	93.05	578.69
	2000 (1400,600)	95.64 (+/- 1.13), 71.14 (+/- 3.60)	94.83, 20.17	42 m 38 s, 2 m 45 s	34.44	370.25
SVM (TF)	100 (70,30)	90.00 (+/- 9.15), 71.43 (+/- 16.90)	100.00, 76.67	1 s, < 1 s	26.00	30.43
	500 (350,150)	96.86 (+/- 1.54), 91.43 (+/- 3.13)	96.67, 82.67	1 s, < 1 s	5.94	16.94
	1000 (700,300)	97.43 (+/- 2.10), 92.29 (+/- 1.71)	92.33, 89.33	3 s , < 1 s	5.57	3.36
	1500 (1050,450)	97.14 (+/- 2.21), 93.05 (+/- 1.76)	94.44, 90.22	18 s, < 1 s	4.40	4.68
	2000 (1400,600)	97.07 (+/- 1.55), 94.29 (+/- 1.72)	95.50, 89.50	1 m 9 s, < 1 s	2.95	6.70

Table 4: Results and comparison of ML algorithm performance. (NN: *Neural Network*; HOG: *Histogram of Oriented Gradients*; TF: *Topological Features*.)

ML Model	# Samples (train, test)	Training accuracy (best, baseline) (%)	Testing accuracy (best, baseline) (%)	Training time (best, baseline)	Improvement of training accuracy (%)	Improvement of testing accuracy (%)
NN (HOG)	100 (70,30)	77.14 (+/- 11.43), 80.00 (+/- 14.57)	86.67,90.00	38 m 32 s, 3 s	-3.57	-3.70
	500 (350,150)	92.29 (+/- 5.72), 94.00 (+/- 4.32)	82.00, 84.00	38 m 8 s, 9 s	-1.82	-2.38
	1000 (700,300)	93.43 (+/- 2.87), 94.43 (+/- 2.51)	90.67, 92.00	47 m 30 s, 24 s	-1.06	-1.45
	1500 (1050,450)	93.43 (+/- 1.78), 93.81 (+/- 1.87)	91.33, 92.67	56 m 51 s, 42 s	-0.41	-1.44
	2000 (1400,600)	93.93 (+/- 2.31), 94.86 (+/- 1.99)	93.67, 95.00	53 m 55 s, 1 m 2 s	-0.98	-1.40
NN (TF)	100 (70,30)	87.14 (+/- 13.48), 82.86 (+/- 15.39)	93.33, 100.00	4 m 16 s, < 1 s	5.17	-6.67
	500 (350,150)	95.43 (+/- 3.66), 93.43 (+/- 4.05)	88.67, 90.00	11 m 46 s, 2 s	2.14	-1.48
	1000 (700,300)	96.43 (+/- 1.84), 94.71 (+/- 2.39)	94.00, 91.67	21 m 49 s, 4 s	1.81	2.55
	1500 (1050,450)	96.86 (+/- 1.76), 95.33 (+/- 1.73)	93.33, 91.33	32 m 23 s, 5 s	1.60	2.19
	2000 (1400,600)	96.21 (+/- 1.75), 95.79 (+/- 1.70)	94.17, 94.50	41 m 13 s, 6 s	0.45	-0.35

[11] Frank Harary. 1994. *Graph Theory.* Reading, MA: Addison-Wesley,.

[12] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd edition).* Springer,

New York.
[13] Christoph Hofer, Roland Kwitt, Marc Niethammer, and Andreas Uhl. 2017. Deep learning with topological signatures. In *Advances in Neural Information Processing Systems*. 1634–1644.
[14] J. Komala Lakshmi and M. Punithavalli. 2009. A survey on skeletons in digital image processing. In *2009 international conference on digital image processing*. IEEE, 260–269.
[15] Louisa Lam, Seong-Whan Lee, and Ching Y Suen. 1992. Thinning methodologies-a comprehensive survey. *IEEE Transactions on pattern analysis and machine intelligence* 14, 9 (1992), 869–885.
[16] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[17] Yuhua Li, Jianwei Zhang, Ming Chen, Haopeng Lei, Guoliang Luo, and Yan Huang. 2018. Shape based local affine invariant texture characteristics for fabric image retrieval. *Multimedia Tools and Applications* (2018), 1–21.
[18] David G Lowe et al. 1999. Object recognition from local scale-invariant features.. In *iccv*, Vol. 99. 1150–1157.
[19] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (2017), e103.
[20] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.
[21] Sriram Pemmaraju and Steven Skiena. 2003. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Cambridge university press.
[22] Urs Ramer. 1972. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing* 1, 3 (1972), 244–256.
[23] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic routing between capsules. In *Advances in neural information processing systems*. 3856–3866.
[24] Panpan Tang and Yuxin Peng. 2017. Exploiting distinctive topological constraint of local feature matching for logo image recognition. *Neurocomputing* 236 (2017), 113–122.
[25] Engin Tola, Vincent Lepetit, and Pascal Fua. 2010. DAISY: An efficient dense descriptor applied to wide-baseline stereo. *IEEE transactions on pattern analysis and machine intelligence* 32, 5 (2010), 815–830.
[26] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. 2014. scikit-image: image processing in Python. *PeerJ* 2 (2014), e453.
[27] Maheswari Visvalingam and James D Whyatt. 1993. Line generalisation by repeated elimination of points. *The cartographic journal* 30, 1 (1993), 46–51.
[28] Douglas Brent West. 2000. *Introduction to graph theory* (2nd ed.). NJ:Prentice-Hall.
[29] Ian H Witten, Eibe Frank, and Mark A Hall. 2011. *Data Mining: Practical machine learning tools and techniques (3rd edition)*. Morgan Kaufmann.
[30] Liping Yang and Guido Cervone. 2019. Analysis of remote sensing imagery for disaster assessment using deep learning: a case study of flooding event. *Soft Computing* (2019), 1–16.
[31] Liping Yang, Alan MacEachren, Prasenjit Mitra, and Teresa Onorati. 2018. Visually-enabled active deep learning for (geo) text and image classification: a review. *ISPRS International Journal of Geo-Information* 7, 2 (2018), 65.

A ABBREVIATIONS

The following abbreviations (ordered alphabetically) are used in this paper:

CNN	Convolutional neural network
CPU	Central processing unit
GNN	Graph neural network
HOG	Histogram of oriented gradients
ML	Machine learning
NN	Neural network
RAM	Random Access Memory
RF	Random forest
SIFT	Scale-invariant feature transform
SURF	Speeded up robust features
SVM	Support vector machine
TF	Topological features

B BRIEF DEFINITIONS OF SOME CONCEPTS IN GRAPH THEORY

In this appendix, we provide brief definitions to some concepts (ordered alphabetically; referenced [4, 5, 11, 21, 28]) in graph theory we used in our graph analysis for topological feature computation.

Algebraic connectivity
: The *algebraic connectivity* of a connected undirected graph is the second smallest eigenvalue of its Laplacian matrix.

Center
: The *center* of a graph is the set of nodes of graph eccentricity equal to the graph radius (i.e., the set of central nodes). A center of a graph is a node with eccentricity equal to the radius. For a general graph, there may be several centers and a center is not necessarily on a diameter. A node n is a central point of a graph if the eccentricity of the node equals the graph radius. The set of all central nodes is called the graph center.

Density
: The *density* for a undirected graph is defined as $D = 2E/N(N-1)$ and for for a directed graph is $D = E/N(N-1)$, where E is the number of edges and N is the number of nodes in the graph.

Diameter
: The *diameter* of a graph is the length of longest path of the shortest paths between any two nodes.

Eccentricity
: The *eccentricity* of a graph node n in a connected graph G is the maximum graph distance between n and any other node m of G. For a disconnected graph, all nodes are defined to have infinite eccentricity. The maximum eccentricity is the graph diameter. The minimum graph eccentricity is called the graph radius.

Edge connectivity
: The *edge connectivity* is equal to the minimum number of edges that must be removed to disconnect G.

Node connectivity
: The *node connectivity* is equal to the minimum number of nodes that must be removed to disconnect G.

Periphery
: The *periphery* of a graph is its subgraph induced by nodes that have graph eccentricities equal to the graph diameter.

Radius
: The *radius* of a graph is the minimum graph eccentricity of any graph node in a graph. A radius of the graph exists only if it has the diameter. A disconnected graph therefore has infinite radius and infinite diameter [28]. (for disconnected graph, instead of setting the infinite radius or diameter, two possible solutions: (1) calculate the sum of the diameter, radius for each component; (2) found the largest component, and then calculate the diameter and radius for the largest component subgraph. In this work, we use the second solution.)

Spectral radius
: The *spectral radius* of a finite graph is defined as the largest absolute value of its graph spectrum, i.e., the largest absolute value of the graph eigenvalues. (The eigenvalues of a graph

Table 5: Hyperparameter settings (from [30]).

ML model	Hyperparameter grid	Combinations	Fits
RF (HOG / TF)	{ 'n_estimators': [10, 20, 30, 40, 50, 60, 70, 75, 80, 85], 'max_depth': [5, 10, 15, 20, 25, 30, 40], 'min_samples_leaf': [2, 4, 6, 8, 10], 'max_features': ['sqrt', 'auto', 'log2', None] }	1400	7000
SVM (HOG / TF)	{ 'kernel': ['linear', 'poly', 'rbf', 'sigmoid'], 'gamma': [10, 1, 0.1, 0.01, 0.001, 0.00001], 'C': [0.1, 1, 10, 100, 1000] }	120	600
NN (HOG / TF)	{ 'alpha': [1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 1e-2]; 'hidden_layer_sizes': [(3,5), (5,10), (9,15), (25,), (50,), (100,), (200,)]; 'solver': ['lbfgs']; 'random_state': [None, 2, 4, 5, 7, 9]; 'max_iter': [100, 200, 300, 500]; 'activation': ['identity', 'logistic', 'tanh', 'relu'] }	4032	20160

are defined as the eigenvalues of its adjacency matrix. The set of eigenvalues of a graph is called a graph spectrum.)

Transitivity

The *transitivity* of a graph is the fraction of all possible triangles present in the graph. It is computed by the equation T = (3 * number of triangles) / number of connected triples of nodes. With this definition, 0 <= T <= 1, and T = 1 if the network contains all possible edges (i.e., the graph is a complete graph).

C COMPUTING ENVIRONMENT

In this appendix, we provide the computing environment that we ran our experiments. We ran all the experiments on a workstation running on Linux (RedHat:enterprise linux:7.6:ga:workstation) with Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz (8 processors, and 4 CPU cores for each processor; 64 GB RAM).

D HYPERPARAMETER SETTINGS AND TUNED RESULTS OF ML ALGORITHMS

In this appendix, we provide some details of the hyperparameter tuning process and results for the three well-known ML algorithms. Table 5 provides the hyperparameter settings including hyperparameter grids for each ML algorithms we tuned (We used the hyperparameter setting shown in [30]), and Table 6 presents the optimized hyperparameters for each ML algorithm and turning time.

In this research, considering the expensive computation of tuning many combinations of hyperparameters, we chose 5-fold cross-validation for the tuning ML modles, even though as introduced in [30], 10-fold is a better setting. We also used all CPU cores (8 * 4 = 32 cores) available on our Linux workstation. Table 5 provides the hyperparameter settings we used to search best parameter for each ML model using Scikit-learn grid search (Grid search is the process of performing hyperparameter tuning in order to determine the optimal values for a given ML model). In Table 5, *Combinations* refers to the number of hyperparameter combinations and *Fits* refers to the number of model fits calculated based on the fold (in our case, the fold number = 5) used for cross-validation and the hyperparameter combinations.

Table 6: Hyperparameter tuning results (RF: *Random Forest*; SVM: *Support Vector Machine*; NN: *Neural Network*; HOG: *Histogram of Oriented Gradients*; TF: *Topological Features*.)

ML model	# Samples (train, test)	Tuning time	Best parameters
RF (HOG)	100 (70,30)	4 m 11 s	{'max_depth': 5, 'max_features': 'auto', 'min_samples_leaf': 2, 'n_estimators': 70}
	500 (350,150)	41 m 39 s	{'max_depth': 30, 'max_features': 'log2', 'min_samples_leaf': 2, 'n_estimators': 85}
	1000 (700,300)	1 h 25 m 20 s	{'max_depth': 30, 'max_features': 'log2', 'min_samples_leaf': 2, 'n_estimators': 75}
	1500 (1050,450)	2 h 29 m 36 s	{'max_depth': 25, 'max_features': 'log2', 'min_samples_leaf': 2, 'n_estimators': 80}
	2000 (1400,600)	3 h 20 m 17 s	{'max_depth': 10, 'max_features': 'log2', 'min_samples_leaf': 2, 'n_estimators': 85}
RF (TF)	100 (70,30)	48 s	{'max_depth': 10, 'max_features': None, 'min_samples_leaf': 2, 'n_estimators': 20}
	500 (350,150)	55 s	{'max_depth': 10, 'max_features': None, 'min_samples_leaf': 8, 'n_estimators': 30}
	1000 (700,300)	1 m 3 s	{'max_depth': 15, 'max_features': None, 'min_samples_leaf': 4, 'n_estimators': 20}
	1500 (1050,450)	1 m 11 s	{'max_depth': 25, 'max_features': None, 'min_samples_leaf': 4, 'n_estimators': 30}
	2000 (1400,600)	1 m 21 s	{'max_depth': 20, 'max_features': None, 'min_samples_leaf': 2, 'n_estimators': 40}
SVM (HOG)	100 (70,30)	9 s	{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
	500 (350,150)	3 m 5 s	{'C': 0.1, 'gamma': 10, 'kernel': 'linear'}
	1000 (700,300)	12 m 4 s	{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
	1500 (1050,450)	25 m 41 s	{'C': 1, 'gamma': 0.01, 'kernel': 'poly'}
	2000 (1400,600)	42 m 38 s	{'C': 0.1, 'gamma': 10, 'kernel': 'poly'}
SVM (TF)	100 (70,30)	1 s	{'C': 1, 'gamma': 10, 'kernel': 'linear'}
	500 (350,150)	1 s	{'C': 1, 'gamma': 10, 'kernel': 'linear'}
	1000 (700,300)	3 s	{'C': 1, 'gamma': 0.01, 'kernel': 'poly'}
	1500 (1050,450)	18 s	{'C': 1, 'gamma': 10, 'kernel': 'linear'}
	2000 (1400,600)	1 m 9 s	{'C': 0.1, 'gamma': 0.01, 'kernel': 'poly'}
NN (HOG)	100 (70,30)	1 h 38 m 32 s	{'activation': 'identity', 'alpha': 0.0005, 'hidden_layer_sizes': (25,), 'max_iter': 300, 'random_state': None, 'solver': 'lbfgs'}
	500 (350,150)	2 h 38 m 8 s	{'activation': 'identity', 'alpha': 0.0005, 'hidden_layer_sizes': (100,), 'max_iter': 300, 'random_state': None, 'solver': 'lbfgs'}
	1000 (700,300)	3 h 47 m 30 s	{'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (200,), 'max_iter': 100, 'random_state': 2, 'solver': 'lbfgs'}
	1500 (1050,450)	4 h 56 m 51 s	{'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (200,), 'max_iter': 100, 'random_state': 5, 'solver': 'lbfgs'}
	2000 (1400,600)	5 h 53 m 55 s	{'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'max_iter': 100, 'random_state': 4, 'solver': 'lbfgs'}
NN (TF)	100 (70,30)	4 m 16 s	{'activation': 'logistic', 'alpha': 5e-05, 'hidden_layer_sizes': (9, 15), 'max_iter': 300, 'random_state': 7, 'solver': 'lbfgs'}
	500 (350,150)	11 m 46 s	{'activation': 'identity', 'alpha': 0.001, 'hidden_layer_sizes': (3, 5), 'max_iter': 100, 'random_state': 4, 'solver': 'lbfgs'}
	1000 (700,300)	21 m 49 s	{'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'max_iter': 500, 'random_state': 2, 'solver': 'lbfgs'}
	1500 (1050,450)	0:32:23	{'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (25,), 'max_iter': 300, 'random_state': 9, 'solver': 'lbfgs'}
	2000 (1400,600)	41 m 13 s	{'activation': 'tanh', 'alpha': 5e-05, 'hidden_layer_sizes': (50,), 'max_iter': 200, 'random_state': 9, 'solver': 'lbfgs'}