Symmetry Detection and Classification in Drawings of Graphs

Felice De Luca[0000−0001−5937−7636], Md Iqbal Hossain(✉)[0000−0001−6212−7638], and Stephen Kobourov[0000−0002−0477−2724]

Department of Computer Science, University of Arizona, USA
{felicedeluca,hossain,kobourov}@cs.arizona.edu

Abstract. Symmetry is a key feature observed in nature (from flowers and leaves, to butterflies and birds) and in human-made objects (from paintings and sculptures, to manufactured objects and architectural design). Rotational, translational, and especially reflectional symmetries, are also important in drawings of graphs. Detecting and classifying symmetries can be very useful in algorithms that aim to create symmetric graph drawings and in this paper we present a machine learning approach for these tasks. Specifically, we show that deep neural networks can be used to detect reflectional symmetries with 92% accuracy. We also build a multi-class classifier to distinguish between reflectional horizontal, reflectional vertical, rotational, and translational symmetries. Finally, we make available a collection of images of graph drawings with specific symmetric features that can be used in machine learning systems for training, testing and validation purposes. Our datasets, best trained ML models, source code are available online.

1 Introduction

The surrounding world contains symmetric patterns in objects, animals, plants and celestial bodies. A symmetric feature is defined by the repetition of a pattern along one of more axes, called *axes of symmetry*. Depending on how the repetition occurs the symmetry is classified as *reflection* when the feature is reflected across the reflection axis, and *translation* when the pattern is shifted in the space. Special cases of reflection symmetries are horizontal (reflective) symmetry when the axis of symmetry is horizontal or a vertical (reflective) symmetry when such axis is vertical. Rotational symmetries occur when the translational axes of symmetry are radial.

Symmetry has been studied in many different fields such as psychology, art, computer vision, and even graph drawing. In psychology, for example, studies on the impact of symmetry on humans show that the vertical symmetry in objects is perceived pre-attentively. A similar study conducted in the context of graph drawing also shows that the vertical symmetry in drawings of graphs is best perceived among all others [8]. In this context, algorithms to measure symmetries in graph drawings have been proposed although it has been shown that these measures do not always agree with what humans perceive as symmetric [34].

Convolutional Neural Networks (CNN) have become a standard image classification technique [18]. CNNs automatically extract features by using information about adjacent pixels to down-sample the image in the first layers, followed by a prediction layer at the end.

Led by the lack of a reliable way to identify a symmetric layout and eventually classify it by the symmetry it contains, in this paper we consider CNNs for the detection and classification of symmetries in graph drawing. Specifically we consider the following two problems: (i) Binary classification of symmetric and non-symmetric layout; and (ii) multi-class classification of symmetric layouts by their type: horizontal, vertical, rotational, translational. In particular, our contributions are as follows:

1. We describe a machine learning model that can be used to determine whether a given drawing of a graph has reflectional symmetry or is not-symmetric (binary classification). This model provides 92% accuracy on our test dataset.
2. We describe a multi-class classification model to determine whether a given drawing of a graph has vertical, horizontal, rotational, or translational symmetry. This model provides 99% accuracy on our test dataset.
3. We make available training datasets, as well as the algorithms to generate them.

The full version of this paper contains more details, figures and tables [7].

2 Related Work

Symmetry detection has applications in different areas such as computer vision, computer graphics, medical imaging, and robotics. Competitions for symmetry detection algorithms have taken place several times; for example, see Liu *et al.* [20]. For reflection and translation symmetries the problem can be interpreted as computing one or more axes of symmetry [17]. In the context of graph drawing, symmetry is one of the main aesthetic criteria [26].

Symmetry detection and computer vision: Detection of symmetry is an important subject of study in computer vision [1, 21, 24]. The last decades have seen a growing interest in this area although the study of bilateral symmetries in shapes dates back to the 1930s [2]. The main focus is on the detection of symmetry in real-world 2D or 3D images. As Park *et al.* [25] point out, although symmetry detection in real-world images has been widely studied it still remains a challenging, unsolved problem in computer vision. The method proposed by Loy and Eklundh [22] performed best in a competition for symmetry detection [20] and is considered a state-of-the-art algorithm for computer vision symmetry detection [6, 25]. Symmetries in 2D points set have also been studied and Highnam [11] proposes an algorithm for discovering mirror symmetries. More recently, Cicconet *et al.* [6] proposed a computer vision technique to detect the line of reflection (mirror) symmetry in 2D and the straight segment

that divides the symmetric object into its mirror symmetric parts. Their technique outperforms the winner of the 2013 competition [20] on single symmetry detection.

Symmetry detection and graphs: In graph theory the symmetry of a graphs is known as automorphism [23] and testing whether a graph has any axial symmetry is an NP-complete problem [3]. A mathematical heuristic to detect symmetries in graphs is given in [9]. Klapaukh [15, 16] and Purchase [26] describe algorithms for measuring the symmetry of a graph drawing. While the first measure analyzes the drawing to find reflection, rotation and translation symmetries, the latter considers only the reflection. Welsh and Kobourov [34] evaluate how well the measures of symmetry agree with human evaluation of symmetry. The results show that in cases where the Klapaukh and Purchase measures strongly disagreed on the scoring of symmetry, human judgment agrees more often with the Purchase metric.

Symmetry detection and machine learning: Convolutional neural networks can be a powerful tool for the automatic detection of symmetries. Vasudevan *et al.* [33] use this approach for the detection of symmetries in atomically resolved imaging data. The authors train a deep convolutional neural network for symmetry classification using 4000 simulated images, 3 convolutional layers, a fully connected layer, and a final 'softmax' output layer on this training dataset. After training over 30 epochs, the authors obtained an accuracy of 85% on the validation set. Tsogkas and Kokkinos [32] propose a learning-based approach to detect symmetry axes in natural images, where the symmetry axes are contours lying in the middle of elongated structures. To the best of our knowledge, there are no prior machine learning approaches for detecting or classifying symmetries in graph drawings.

Neural networks for image classification and detection: Convolutional Neural Networks (CNNs) are standard in image recognition and classification, object detection, and video analysis. The Mark I Perception machine was the first implementation of the perceptron algorithm in 1957 by Rosenblatt [27]. Widrow and Hoff proposed a mutlilayer perceptron [35]. Back-propagation was introduced by Rumelhart *et al.* [28]. LeNet-5 [19] was deployed for zip code and digit recognition. In 2012, Alex Krizhevsky [18] introduced CNNs with AlexNet. Szegedy *et al.* [14] introduced GoogLeNet and the Inception module. Other notable developments include VGGNet [30] and residual networks (ResNet) [10].

3 Background and Preliminaries

In this section we give a brief overview of machine learning in the context of our experiments. We also attempt to clarify some of the terminology we use throughout the paper, focusing in particular on *Deep Neural Networks* and *Convolutional Neural Networks*.

A deep neural network is made of several layers of neurons. Information flows through a neural network in two ways: via the *feedforward network* and via *backpropagation*. During the training phase, information is fed into the network via the input units, which trigger the layers of hidden units, and these in turn arrive at the output units. This common design is called a *feedforward network*. Not all units fire all the time. Each unit receives inputs from the units of the previous layer, and the inputs are multiplied by the weights of the connections they travel along. Every unit adds up all the inputs it receives in this way and if the sum exceeds a certain threshold value, the unit fires and triggers the units it is connected to in the next layer.

Importantly, there is a feedback process called *backpropagation* that can be used to improve the weights. This involves the comparison of the output the network produces with the output it was meant to produce, and using the difference between them to modify the weights of the connections between the units in the network, working from the output units, through the hidden units, and to the input units. Over time, backpropagation helps the network to "learn," reducing the difference between actual and intended outputs.

Convolutional neural network (CNN) are used mainly for image data classification where intermediate layers and computations are a bit different then fully connected neural networks. Each pixel of input image is mapped with a neuron of the input layer. Output neurons are mapped to target classes. Figure 1 shows a simple CNN architecture. Different types of layers in a typical CNN include:

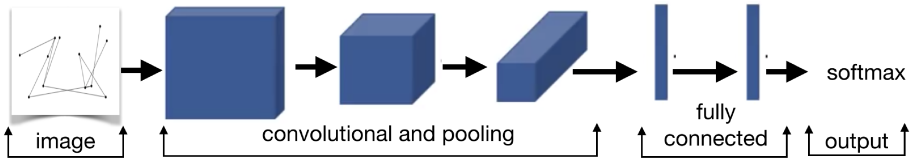


Fig. 1: A typical convolutional neural network.

- *convolution layer (convnet)*: in this layer a small filter (usually 3×3) is taken and moved over the image. Applying filters in the layer helps to detect low and high level features in the image so that spatial features are preserved in the layer. The convolutional layer helps to reduce the number of parameters compared to a fully connected layer. Keeping the same set of filters helps to share parameters and sparsity helps to further reduce the parameters. For example, in a 3×3 filter every node in the next layer is only connected to 9 nodes in the previous layer. This sparse connection helps to avoid overfitting.
- *activation layer*: this layer applies an activation function from the previous layer. Example functions include ReLU, tanh and sigmoid.

– *pooling*: the pooling layer is used to reduce size of the convnet. Filter size f, stride s, padding p are used as parameters of the pooling layer. Average pooling or max pooling are the standard options. After applying the pooling to a given image shape $(N_h \times N_w \times N_c)$, it turns into $\lfloor \frac{N_h - f}{s} + 1 \rfloor \times \lfloor \frac{N_w - f}{s} + 1 \rfloor \times N_c$.
– *Fully Connected Layer* (FCL): a fully connected layer creates a complete bipartite graph with the previous layer. Adding a fully-connected layer is useful when learning combinations of non-linear features.

We now review some common machine learning terms. *Training loss* is the error on the training set of data, and *validation loss* is the error after running the validation set of data through the trained network. Ideally, train loss and validation loss should gradually decrease, and training and validation accuracy should increase over training epochs. The *training set* is the data used to adjust the weights on the neural network. The *validation set* is used to verify that increase in accuracy over the training data actually yields an increase in accuracy. If the accuracy over the training data set increases, but the accuracy over the validation data decreases, it is a sign of *overfitting*. The *testing set* is used only for testing the final solution in order to confirm the actual predictive power of the network. A *confusion matrix* is a table summarizing the performance in classification tasks. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class. The *precision* p represents how many selected item are relevant and *recall* r represents how many relevant items are selected. $F1$-*score* is measured by the formula $2 * \frac{r*p}{p+r}$.

4 Datasets

In this section we describe how we generated datasets for our machine learning systems. To the best of our knowledge, there is no dataset of images suitable for training machine learning systems for symmetry detection in graph drawings. Our dataset contains images that feature different types of symmetries, including reflection, translation or rotation symmetries and variants thereof. An overview all types of layouts is given in Fig. 2.

We started with a dataset of simple symmetric images and inspected the results trying to identify which characteristic of the layout leads to its classification as symmetric or not symmetric. If we observed a characteristic in the symmetric layouts we generated non symmetric layouts that expose it and symmetric layouts without it. Then we fed them to the system for the classification. In case of inaccurate results we included the new layouts (that we call *breaking instances* of the dataset) in the training system and repeated the process until we could not identify any other specific feature.

In order to distinguish inputs of different sizes, we refer to layouts in our dataset as *small* or *large* based on the number of vertices, $|V|$. A small layout has $|V| \in [5, 8]$ while a large layout has $|V| \in [10, 20]$. The number of edges is a random integer $|E| \in [|V|, \lfloor 1.2 * |V| \rfloor]$. The layouts included in the global dataset used for all experiments can be summarized as follows:

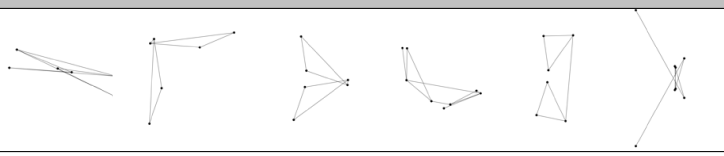
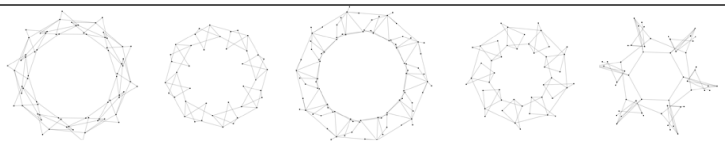
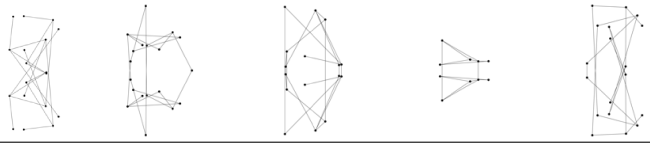
Name of Set	Samples	Size
Small Sym		10000
Small NonSym		10000
Reflectional Large		10000
NonSym Large		10000
Rotational Large		8000
Translational Large		8000
Horizontal Large		8000
Vertical Large		8000

Fig. 2: Examples of the different layout instances in our dataset.

- *SmallSym*: small reflective symmetric layout
- *SmallNonSym*: non symmetric generated from SmallSym with random node positions
- *ReflectionalLarge*: large reflective symmetric layouts with random axis of symmetry
- *NonSymLarge*: non symmetric generated from ReflectionalLarge layouts
- *HorizontalLarge*: large reflective symmetric layouts with a 0 degree axis of symmetry
- *VerticalLarge*: large reflective symmetric layouts with a 90 degree axis of symmetry
- *RotationalLarge*: rotational symmetric with random axes between 4 and 10
- *TranslationalLarge*: translational symmetric translated along x-axis

In the remainder of this section we discuss how we generated our layouts and the process that led to them.

4.1 Reflectional Layout Generation

A reflectional symmetric layout may expose different characteristics such as "parallel lines" orthogonal to the axis of symmetry and edge crossings on the axis of symmetry.

The generation procedure for symmetric graphs and layouts thereof differs slightly depending on the type of symmetry we attempt to capture.

We used the procedure for generating a graph and a reflectional symmetric layout with the "parallel lines" feature following the algotithm in [8] as follows. Given a graph with $\frac{n}{2}$ vertices, called a *component*, we assign to each vertex of the component positive random coordinates. Then we copy this component and replace the x-coordinates of each vertex with the negative value of the original. This results in a layout with two disjoint components that are then connected by a random number of edges in $[1, \lfloor |V|/3 \rfloor]$ selecting random vertices in one component and connecting them to their corresponding vertices in the other component. This results in layouts with vertical axis of symmetry; see Fig. 3(b). To create layouts with horizontal axis of symmetry we add a 90 degree rotation; see Fig. 3(a).

The procedure for generating a graph and a reflectional symmetric layout without the "parallel lines" feature is described in Algorithm *SymGG*. This algorithm gives an overview on how to create the symmetric versions with the different features. In the following we explain how we defined *SymGG* based on experimental improvements of our dataset. Given a symmetric graph with n vertices by Algorithm *SymGG*, we create a non-symmetric layout by assigning to each vertex of the input graph any random y-coordinate and a positive random x-coordinate to the vertices with identifier $< \frac{n}{2}$ and a negative random x-coordinate, otherwise.

To create reflectional symmetric layouts, instead, if a vertex with identifier $i < \frac{n}{2}$ gets coordinates (x_r, y_r) then the vertex with identifier $i_c = i + \frac{n}{2}$ gets assigned coordinates $(-x_r, y_r)$. If the graph has an odd number of vertices then

the vertex with identifier $n-1$ gets $x = 0$. Note that, by construction, the resulting layouts have a vertical axis of symmetry; see Fig. 3(e). To create layouts with horizontal axis of symmetry we add a 90 degree rotation; see Fig. 3(f).

Algorithm SymGG(n,m): Symmetric graph generation with n vertices and m edges

1: define $G = (V, E)$ where $|V| = n$ with id $[0, n-1]$ and $|E| = 0$
2: add m edges to G selecting one or more edge types from [3-6] and continuing with steps [7-12]
3: for a random edge choose random integers u, v in $[0, n-1]$ such as $(u, v) \notin E$;
4: for a random edge that does not cross the axis of reflection choose random integers u, v in $[0, \lfloor n/2 \rfloor - 1]$ such as $(u, v) \notin E$;
5: for parallel edge feature choose random integer u in $[0, \lfloor n/2 \rfloor - 1]$ and $v = u + \lfloor n/2 \rfloor$ such as $(u, v) \notin E$;
6: for crossing edge feature choose random integer u in $[0, \lfloor n/2 \rfloor - 1]$ and v in $[n/2, n-1]$ such as $(u, v) \notin E$;
7: Generate the symmetric edge (u_sym, v_sym) of (u, v)
8: $u\_sym = u \mp \lfloor n/2 \rfloor$ if $u \gtrless \lfloor n/2 \rfloor$
9: $v\_sym = v \mp \lfloor n/2 \rfloor$ if $v \gtrless \lfloor n/2 \rfloor$
10: $u\_sym = u$ if n is odd and $u = n - 1$
11: $v\_sym = v$ if n is odd and $v = n - 1$
12: add (u, v) and (u_sym, v_sym) to E

4.2 Dataset Definition

Here we describe the process that led to us to the dataset of reflectional symmetric layouts.

To this aim we generated the SmallSym, SmallNonSym, NonSymLarge and ReflectionalLarge layouts.

First improvement: At first, we trained our system with the reflective symmetric layouts and random layouts generated using the approach in [8] as described above.

Observations: Using this simple dataset we observed that the system could always classify the layouts correctly for any of the used layouts.

Layouts characteristic: Analyzing the used dataset we observed that the generation algorithm used gives symmetric layout for reflective symmetry with a clear symmetric feature that is 'parallel lines' orthogonal to the reflection axis. These lines separate two identical but reflected subcomponents, as Fig. 3(a-b) show.

Breaking layout: After identifying the 'parallel lines' feature, we generated non-symmetric layouts with the same feature. These layouts were created starting from the symmetric layouts and then assigning random positions to the vertices not linked to the parallel edges; an example of random layout with parallel edges is shown in Fig. 4b. Without re-training the system, these layouts are misclassifed as symmetric, breaking the previously built model.

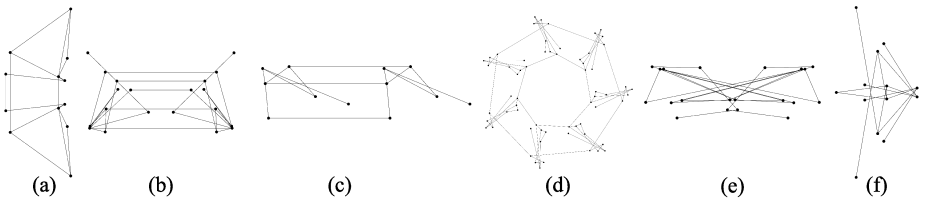


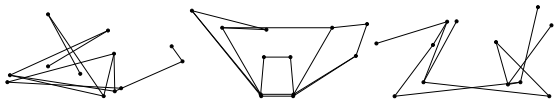
Fig. 3: Symmetric layouts in the dataset: (a) Horizontal, (b) Vertical, (c) Translational, (d) Rotational, (e) Vertical without parallel lines, (f) Horizontal without parallel lines.

Second improvement: Here we added to our dataset the breaking instances of the previous model and new symmetric layouts that do not show the 'parallel lines' feature. The parallel lines of a symmetric layouts are given by vertices that are connected to their reflected copy (since they share either the x or y coordinate in the space). The new layouts we generated have the two subcomponents not only connected by edges between a vertex and his reflected copy but also by edges connecting a random vertex of one component to a random vertex of the other (and viceversa to keep the symmetry). These edges generate crossings on the axis of symmetry of the symmetric layout, instead of the parallel lines. Pseudocode for the symmetric graph generation Algorithm SymGG (with even number of edges as input) can be found above.

Analogously we generated some random layouts that show the same feature, starting from a symmetric layout with non-parallel edges and shuffling the position of the vertices not connected to such edges. Figure 3(e) illustrates and example of symmetric layout with crossings while Fig. 4(c) depicts a non symmetric layout with crossings.

Observations: Training the system with these new layouts we obtained good results on all layouts, including those misclassified in the previous setup.

Breaking instance: Inspecting the current dataset we identified another characteristic of the current symmetric layouts: an even number of vertices. We then generates symmetric layouts with an odd number of vertices. The generation algorithm for these layouts is given in Algorithm SymGG. Again, without training, the the current system fails on such layouts misclassifying them as a



(a) (b) (c)

Fig. 4: Non symmetric layouts in the dataset: (a) Random, (b) with Parallel Lines, (c) with Crossings

non-symmetric. Further, we observed that rotating the symmetric layouts also makes our machinery fail.

Final improvement: Here we added to our dataset instances with odd number of vertices for both symmetric and non symmetric layouts. We also added instances rotated by a random angle between 0 and 360. Since we could not find further breaking instance for this dataset, we used it for our experiments.

4.3 Other symmetric layouts

In addition to the instances above we generated the translational layouts and rotational layouts using the algorithm in [8], as follows.

To create translational symmetric layouts we use the same process of generation of reflectional symmetric layout with parallel edges above but instead of taking the negative value of the x-coordinate of the copied component we shift each component by a predefined value δ. If a vertex in the given component gets coordinates (x, y) then the vertex in the copied component is assigned coordinates $(x - \delta, y)$; see Fig. 3(c).

The generation process for rotational symmetric layouts is different, since the number of vertices depends on the number of symmetric axes. To generate such layouts we start from a given graph component with n vertices and then we select a random number of radial symmetric axes in the range $[4, 10]$. After assigning a random position to the vertices of the component we copy and shift it over the reflection axes. Then we choose two random vertices in the component and use them to connect pairs of rotationally consecutive components; see Fig. 3(d).

5 Experimental Setup

Our images are in black and white with a size of 200×200 pixels. We use 1 pixel for the edge width and 3×3 pixels for a vertex. We configured our system with the following settings: 1 grayscale channel, with resealing by $1/255$, batch size 16 and number of epochs 20. In all of our experiment we use 80% of our data as training set, 10% as validation set, and 10% as test set. Test sets are never used in during training, those are reserved for computing the final accuracy. During training, in every epoch we check the validation accuracy and save the

best trained model as checkpoint. The best trained model is used on the final test set.

Since images of graph drawings have different features than that of real-world images (e.g., textures and shapes), we tested different popular CNN architectures with same parameter settings.

Name	parameters	layers	references	our training time (h)
ResNet50	23.59M	177	[10]	15.25
MobileNet	3.23M	93	[12]	6.22
MobileNetV2	2.26M	157	[29]	8.36
NASNetMobile	4.27M	771	[36]	5.79
NASNetLarge	84.93M	1041	[36]	10.21
VGG16	107.01M	23	[30]	24.24
VGG19	112.32M	26	[4]	25.32
Xception	20.87M	134	[5]	19.59
InceptionResNetV2	54.34M	782	[31]	15.18
DenseNet121	7.04M	429	[13]	20.11
DenseNet201	18.32M	709	[13]	28.49

Table 1: Overview of the CNN models used in the experiment.

We use CNN architectures from the Keras implementation; Keras is a high-level API of Tensorflow that supports training with multiple CPUs[1]. For our experiments, we used the High Performance Computing system at the University of Arizona. Specifically, training was done on 28 CPUs, each with Intel Xeon 3.2GHz processor and 6GB of memory. Training time for the different models ranged from 6 to 29 hours; see Table 1.

6 Detecting reflectional symmetry

Small Binary Classification **(SPBC) Experiment:** In this experiment we test how accurately we can distinguish between drawings of graphs with reflectional symmetry and ones without. We use a binary classifier trained on *SmallSym* and *SmallNonSym* instances from our dataset; see Fig. 2. We use the *InceptionResNet* CNN model with 12000 images for training, 2000 images for validation, and 2000 image for testing. The model achieves 92% accuracy. We evaluated several different models before settling on *InceptionResNet*; see the full paper for more details [7].

We cross-validate our results with two earlier metrics specifically designed to evaluate the symmetry in drawing of graphs, namely the Purchase metric [26] and the Klapaukh metric [15]. These two metrics were not designed for binary

[1] `https://github.com/keras-team/keras/tree/master/keras/applications`

classification, but given a graph layout they provide a score in the range $[0, 1]$. We interpret a score of ≥ 0.5 as a vote for "symmetry" and a score of < 0.5 as a vote of "no symmetry." We can now compare the performance of our CNN model against those of the Purchase metric and the Klapaukh metric on the same set of 2000 test images. We report accuracy, precision, recall and F1-score in Table 2. We can see that while the two older metrics perform well, the CNN is better in all aspects (except recall, where the Purchase metric is .01% better).

Model	Accuracy	Precision	recall	F1-Score
Purchase [26]	82%	0.67	0.96	0.79
Klapaukh [15]	82%	0.80	0.86	0.83
InceptionResNet	92%	0.90	0.95	0.93

Table 2: Comparison between the CNN model and existing symmetry metrics.

Training loss, validation loss, training accuracy and validation accuracy for our *E*xperiment SPBC are shown in the full version of the paper [7].

7 Detecting different types of symmetries

Multi-class symmetric layouts classification (LHVRT) Experiement: In this experiment we test how accurately we can distinguish between drawings of graphs with different types of symmetries. We use a multi-class classifier trained on several types of symmetries: Horizontal, Vertical, Rotational and Translational. Recall that Horizontal and Vertical are special cases of reflection symmetry, where the axis of reflection is horizontal or vertical, respectively.

We train the CNN with *HorizontalLarge*, *VerticalLarge*, *RotationalLarge*, and *TranslationalLarge* instances from our dataset; see Fig. 2.

We use the *ResNet50* CNN model with 16000 images for training, 2000 images for validation, and 4280 image for testing. The model achieves 99% accuracy. Table 3 shows the corresponding confusion matrix. We evaluated several different models before settling on *ResNet50*. Training loss, validation loss, training accuracy and validation accuracy for our *E*xperiment LHVRT are shown in the full version of the paper, where more results and discussion thereof can also be found [7].

8 Conclusions

In the experiments above we achieved high accuracy for both detection and classification. Compared to existing evaluation metrics for symmetric layout we observed that our machinery outperforms the mathematical formulae proposed when used as classifiers.

	HorizontalLarge	RotationalLarge	TranslationalLarge	VerticalLarge
HorizontalLarge	1280	0	0	0
RotationalLarge	0	800	0	0
TranslationalLarge	0	0	798	2
VerticalLarge	0	0	1	1599

Table 3: Confusion matrix from *ResNet50*. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class.

Note, however, that there are many limitations to consider. First of all, we generated all the datasets and have not tested the models on layouts obtained from other layout algorithms. Further, the graphs we used are small and we have not confirmed how well humans agree with the decisions of the machine learning system. Finally, the two tasks we performed are limited in power, and we do not yet have a model that can accurately predict whether a graph drawing is symmetric or not, or which of two drawings of the same graph is more symmetric.

Nevertheless, we believe our dataset can be useful for future experiments and our initial results on limited tasks indicate that a machine learning framework can be useful for symmetry detection and classification. Our dataset, models, results details can be found in `https://github.com/enggiqbal/mlsymmetric`

Acknowledgement
This work is supported in part by NSF grants CCF-1740858, CCF-1712119, DMS-1839274, and DMS-1839307. This experiment uses High Performance Computing resources supported by the University of Arizona TRIF, UITS, and RDI and maintained by the University of Arizona Research Technologies department.

References

1. Atallah, M.J.: On symmetry detection. IEEE Transactions on Computers **C-34**(7), 663–666 (July 1985). https://doi.org/10.1109/TC.1985.1676605
2. Birkhoff, G.D.: Aesthetic measure. Cambridge, Mass. (1932)
3. Brendan Manning, J.: Geometric symmetry in graphs. ETD Collection for Purdue University (05 1990)
4. Chen, L., Zhang, H., Xiao, J., Nie, L., Shao, J., Liu, W., Chua, T.S.: SCA-CNN: Spatial and channel-wise attention in convolutional networks for image captioning. In: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017. vol. 2017-January, pp. 6298–6306 (2017). https://doi.org/10.1109/CVPR.2017.667
5. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. In: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017. vol. 2017-January, pp. 1800–1807 (2017). https://doi.org/10.1109/CVPR.2017.195
6. Cicconet, M., Birodkar, V., Lund, M., Werman, M., Geiger, D.: A convolutional approach to reflection symmetry. Pattern Recognition Letters **95** (09 2016). https://doi.org/10.1016/j.patrec.2017.03.022

7. De Luca, F., Hossain, M.I., Kobourov, S.: Symmetry detection and classification in drawings of graphs. arXiv preprint arXiv:1907.01004 (2019)
8. De Luca, F., Kobourov, S., Purchase, H.: Perception of symmetries in drawings of graphs. In: Biedl, T., Kerren, A. (eds.) Graph Drawing and Network Visualization. pp. 433–446. Springer International Publishing, Cham (2018)
9. Fraysseix, H.d.: An heuristic for graph symmetry detection. In: Proceedings of the 7th International Symposium on Graph Drawing. pp. 276–285. GD '99, Springer-Verlag, London, UK, UK (1999)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. vol. 2016-December, pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90
11. Highnam, P.T.: Optimal algorithms for finding the symmetries of a planar point set. Tech. Rep. 5, Carnegie Mellon University, Pittsburgh, PA (August 1986). https://doi.org/10.1016/0020-0190(86)90097-9
12. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861 (2017), `http://arxiv.org/abs/1704.04861`
13. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017. vol. 2017-January, pp. 2261–2269 (2017). https://doi.org/10.1109/CVPR.2017.243
14. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. 32nd International Conference on Machine Learning, ICML 2015 **1**, 448–456 (2015)
15. Klapaukh, R.: An Empirical Evaluation of Force-Directed Graph Layout. Ph.D. thesis, Victoria University of Wellington (2014)
16. Klapaukh, R., Marshall, S., Pearce, D.: A symmetry metric for graphs and line diagrams. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 10871 LNAI, pp. 739–742. Springer (2018). https://doi.org/10.1007/978-3-319-91376-6_71
17. Kokkinos, I., Maragos, P., Yuille, A.: Bottom-up amp;amp; top-down object detection using primal sketch features and graphical models. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06). vol. 2, pp. 1893–1900 (June 2006). https://doi.org/10.1109/CVPR.2006.74
18. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. vol. 2, pp. 1097–1105 (2012)
19. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2323 (1998). https://doi.org/10.1109/5.726791
20. Liu, J., Slota, G., Zheng, G., Wu, Z., Park, M., Lee, S., Rauschert, I., Liu, Y.: Symmetry detection from realworld images competition 2013: Summary and results. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 200–205 (2013)
21. Liu, Y., Hel-Or, H., Kaplan, C.S., Van Gool, L.: Computational Symmetry in Computer Vision and Computer Graphics. now (2010)
22. Loy, G., Eklundh, J.O.: Detecting symmetry and symmetric constellations of features. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) Lecture Notes in Computer

Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 3952 LNCS, pp. 508–521. Springer Berlin Heidelberg, Berlin, Heidelberg (2006). https://doi.org/10.1007/11744047_39
23. Lubiw, A.: Some NP-Complete Problems Similar to Graph Isomorphism. SIAM Journal on Computing **10**(1), 11–21 (1981). https://doi.org/10.1137/0210002
24. Mitra, N.J., Pauly, M., Wand, M., Ceylan, D.: Symmetry in 3d geometry: Extraction and applications. Computer Graphics Forum **32**(6), 1–23 (2013). https://doi.org/10.1111/cgf.12010
25. Park, M., Lee, S., Chen, P.C., Kashyap, S., Butt, A.A., Liu, Y.: Performance evaluation of state-of-the-art discrete symmetry detection algorithms. In: 26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR. pp. 1–8 (June 2008). https://doi.org/10.1109/CVPR.2008.4587824
26. Purchase, H.C.: Metrics for graph drawing aesthetics. Journal of Visual Languages and Computing **13**(5), 501–516 (2002). https://doi.org/10.1016/S1045-926X(02)90232-6
27. Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review **65**(6), 386–408 (1958). https://doi.org/10.1037/h0042519
28. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**(6088), 533–536 (1986). https://doi.org/10.1038/323533a0
29. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: MobileNetV2: Inverted Residuals and Linear Bottlenecks. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. pp. 4510–4520 (2018). https://doi.org/10.1109/CVPR.2018.00474
30. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556 (2014), `http://arxiv.org/abs/1409.1556`
31. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-ResNet and the impact of residual connections on learning. In: 31st AAAI Conference on Artificial Intelligence, AAAI 2017. pp. 4278–4284 (2017)
32. Tsogkas, S., Kokkinos, I.: Learning-based symmetry detection in natural images. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) Computer Vision – ECCV 2012. pp. 41–54. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
33. Vasudevan, R.K., Dyck, O., Ziatdinov, M., Jesse, S., Laanait, N., Kalinin, S.V.: Deep Convolutional Neural Networks for Symmetry Detection. Microscopy and Microanalysis **24**(S1), 112–113 (2018). https://doi.org/10.1017/s1431927618001058
34. Welch, E., Kobourov, S.: Measuring Symmetry in Drawings of Graphs. Computer Graphics Forum **36**(3), 341–351 (2017). https://doi.org/10.1111/cgf.13192
35. Widrow, B., Lehr, M.A.: 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation. Proceedings of the IEEE **78**(9), 1415–1442 (1990). https://doi.org/10.1109/5.58323
36. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)

Appendix

9 Discussion

We performed further experiments to test the behavior of the machine learning system on slightly larger graphs and with slightly more difficult tasks. We report on two such experiments, LNBC and LRefRotTra, below.

9.1 Experiment LNBC (*ReflectionalLarge*, *NonSymLarge*)

In this experiment we use more complex input instances when detecting symmetries. The dataset *ReflectionalLarge* and *NonSymLarge* are used as symmetric and non-symmetric in this experiment with 10000 samples; see Fig. 2. We use 80% of our data as training set, 10% as validation set, and 10% as test set. Note that *ReflectionalLarge* dataset combines all types of reflection symmetries, including horizontal reflection, vertical reflection and arbitrary axis reflection. This makes the set of symmetric instances more varied then when considering only one type of symmetry. Further, *NonSymLarge* contains more complex non-symmetric instances, where starting from a symmetric layout a few vertices are slightly perturbed in order to break the symmetry. This task is clearly harder and accuracy decreases to 78%; see Table 5.

The experiment above motivates three more focused experiments that we use to identify the nature of the difficulty of the LNBC task .

- **Experiment LHnonSym**: tests whether the model can distinguish between complex non-symmetric (dataset *NonSymLarge*) and only horizontal symmetric samples (dataset *HorizontalLarge*).
- **Experiment LVnonSym**: tests whether the model can distinguish between complex non-symmetric (dataset *NonSymLarge*) and only vertical symmetric samples (dataset *VerticalLarge*).
- **Experiment LHVSym**: tests whether the model can distinguish between horizontal symmetric (dataset *HorizontalLarge*) and only vertical symmetric samples (dataset *VerticalLarge*).

Table 5 shows that several of models achieve 100% accuracy for all 3 of these experiments (LHnonSym, LVnonSym and LHVSym). This provides a possible explanation for the low accuracy of the LNBC experiment: the machine learning algorithms are struggling to distinguish symmetric from non-symmetric layouts when both the symmetric instances are more complex (different types of symmetries) and when the non-symmetric instances are also more complex (different types of non-symmetric layouts).

9.2 Experiment LRefRotTra (*ReflectionalLarge*, *RotationalLarge*, *TranslationalLarge*)

We next conducted an experiment to detect the type of symmetry in a given layout, from the possible options: reflectional, rotational, and translational. Note

that in this experiment we do not distinguish among the various types of reflectional symmetry (horizontal, vertical, arbitrary axis). That is, the reflectional layouts include vertical, horizontal, and reflectional with random angle of rotation samples. From the total of 24720 instances in these three datasets, we choose 80% for training, 10% for validation, and 10% for testing; see the *ReflectionalLarge*, *RotationalLarge* and *TranslationalLarge* rows in Fig. 2.

The best performing models achieve 69% accuracy, which indicates difficulty in distinguishing the different types of symmetries. In particular, the confusion matrix in Table 4 shows that the translational symmetric instances are incorrectly detected as reflectional symmetric instances.

	ReflectionalLarge	RotationalLarge	TranslationalLarge
ReflectionalLarge	872	0	0
RotationalLarge	0	800	0
TranslationalLarge	800	0	0

Table 4: Confusion matrix of *Experiment LRefRotTra*. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class. Note that the translational symmetric instances are incorrectly detected as reflectional symmetric instances.

10 Experimental statistics

In this section we present some statistics of training progress of different models. Figure 5 show training loss, validation loss, training accuracy and validation accuracy of different CCN architectures for *Experiment SPBC*. For each graphic, the x-axis represents the epochs and the y-axis represents the value of loss or accuracy, depending on the color of the line (see legend). Overall, we ovserve that InceptionResNetV2 has the best behavior showing decreasing loss and increasing accuracy. Figure 6 shows similar statistics for *Experiment LHVRT* where only two models converge fast, namely ResNet50, InceptionResNetV2.

We summarize results of all experiments in the Table 5 where training and validation accuracy are reported.

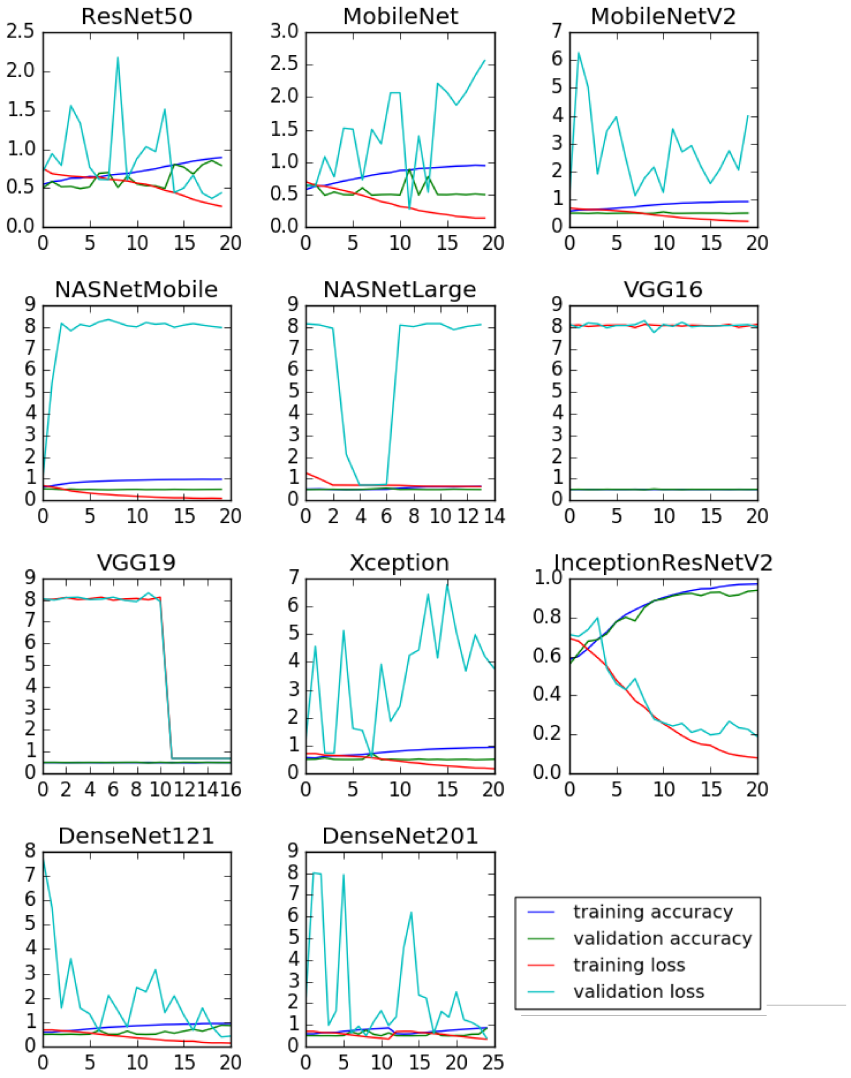


Fig. 5: Training loss, validation loss, training accuracy and validation accuracy of different CCN architectures for binary classification. The x-axis represents epochs and the y-axis represents loss or accuracy. InceptionResNetV2 shows the correct behavior with decreasing losses increasing accuracy.

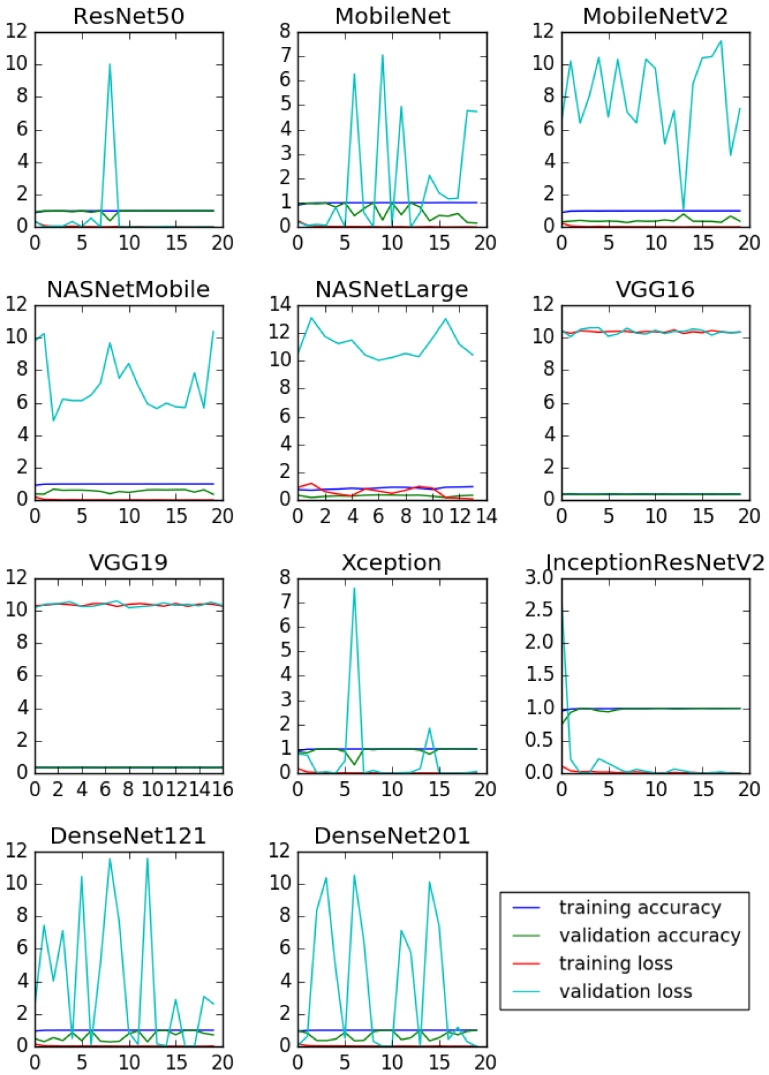


Fig. 6: Training loss, validation loss, training accuracy and validation accuracy of different CCN architectures for multi-class classification.

model	SPBC		LNBC		LHnonSym		LHVSym		LVnonSym		LHVRT		LRefRotTra	
	tracc	vacc	tracc	vacc	tracc	vacc	tracc	vacc	tracc	vacc	tracc	vacc	tracc	vacc
ResNet50	0.89	0.85	0.89	0.76	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.68
MobileNet	0.95	0.89	0.96	0.72	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.69
MobileNetV2	0.92	0.55	0.93	0.42	1.0	1.0	1.0	1.0	1.0	0.89	1.0	0.8	1.0	0.68
NASNetMobile	0.97	0.52	0.98	0.63	1.0	0.59	1.0	1.0	1.0	0.53	1.0	0.68	1.0	0.66
NASNetLarge	0.64	0.56	0.87	0.67	0.99	0.68	1.0	0.92	1.0	0.54	0.97	0.38	1.0	0.67
VGG16	0.51	0.52	0.47	0.41	0.56	0.58	0.45	0.47	0.51	0.53	0.36	0.38	0.51	0.33
VGG19	0.51	0.51	0.46	0.42	0.56	0.58	0.45	0.46	0.51	0.53	0.36	0.37	0.51	0.34
Xception	0.94	0.72	0.97	0.71	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.68
InceptionResNetV2	0.97	0.94	0.99	0.78	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.69
DenseNet121	0.95	0.87	0.96	0.62	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.69
DenseNet201	0.86	0.83	0.89	0.69	1.0	0.99	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.68

Table 5: Training accuracy (tracc) and validation accuracy (vacc) achieved by different models for the different tasks, at a glance.