Universiteit Leiden
The Netherlands

Opleiding Informatica & Economie

Automated Rotational and Localised Symmetry Detection from Global Reflection Symmetries from 2d Images

Koen Ponse

Supervisors:
Bas van Stein & Anna V. Kononova

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

12/07/2021

Abstract

Automated symmetry detection is still a difficult task in 2021. However, it has applications in computer vision, and it also plays a large part in understanding art. This thesis will focus on aiding the latter by comparing different state-of-the-art automated symmetry detection algorithms. Furthermore, we propose post-processing improvements to an algorithm to find more localised symmetries in images, improve the selection of detected symmetries and detect rotational symmetries. In order to detect rotational symmetries, we contribute a machine learning model which detects rotational symmetries based on provided reflection symmetry axis pairs. We conclude that we have successfully extended an algorithm meant to detect global reflection symmetries to detect localised symmetries and rotational symmetries. Some detection methods did not perform well on our data set.

Contents

1 Introduction

1.1 Problem Description

In everyday life, humans and animals use their innate perception of symmetry to identify objects quickly. This very same application could be useful in computer vision. Symmetry also plays a significant role in art as it is one of the main art principles with which artists create the composition of an artwork [7]. Automated symmetry detection could therefore improve various computer tasks and also aid art scientists and cultural heritage projects. This thesis aims to help the latter by analysing the current state-of-the-art automated symmetry detection algorithms and propose various ideas to improve them or make more use of these algorithms.

Ornaments & patterns online archive Ornamika [1] has kindly provided us with their database containing diverse artistic images. Labelling these images with their symmetries and symmetry types will help organise their database. This will, in turn, aid them in their work and aid other designers who use their database. However, due to a large number of images, an automated approach for labelling the images is required. The Ornamika data set is different from most other data sets on which existing algorithms have been tested. Most papers test their algorithms on data sets containing real-world photos. Such photos often contain complex backgrounds, lighting and depth. In contrast, the Ornamika data set is mainly comprised of ornaments on a 2-d plane. Often the image features a flat textile art piece without other irrelevant background objects. When an image in the Ornamika data set is not a flat surface, it is often an object which has been photographed *en face* in good lighting conditions. Since other papers have used a different kind of data, it is currently unknown which algorithms perform best on a data set like the one from Ornamika. Additionally, it is difficult to predict whether existing algorithms can be improved to yield better results.

Over the years, many papers [2, 16, 4, 19, 11] have been published proposing various symmetry detection methods. Three different symmetry detection competitions have been held with increased complexity in each competition. Further details about these competitions are shared in Section 2. Despite the amount of work in the field, automated symmetry detection does still prove to be a difficult task, as we can conclude based on the last symmetry detection competition, which was held in 2017 [9]. Their conclusion states that an old algorithm of 2006 was still competitive.

1.2 Contribution

We performed a literature study on the current state-of-the-art algorithms. We then compared different algorithms on the Ornamika data set to determine which algorithm works best on this data set. We selected two different algorithms to include in our experiments. One for detecting medial axis symmetries [25] and one for detecting global reflection symmetries [8] (see Section 1.3 for definitions).
We propose various improvements to the latter. First of all, we improve the selection of discovered symmetries to lower the false-positive detection rate. Furthermore, we utilise this algorithm, meant for detecting global symmetries, to find more localised symmetries in our images. Lastly, we use the algorithm in order to detect rotational symmetries. We propose a set of rules with parameters with which detected reflection symmetries can determine the existence of a rotational symmetry.

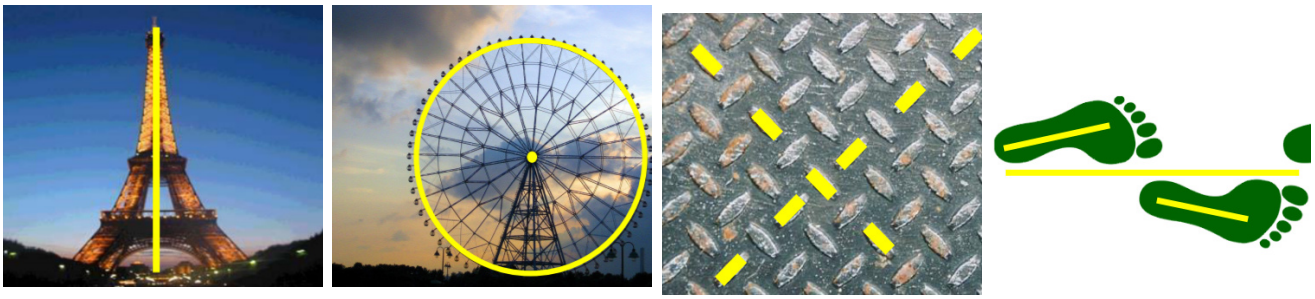
We subsequently replace this rule-based method with a machine learning method and show that Elawady's reflection symmetry detection algorithm can detect rotational symmetries using our model with an accuracy of over 90% on our test set. We note that, in theory, any reflection symmetry detection algorithm will be able to use our trained model to detect rotational symmetries, as its input only requires a set of coordinates of two lines and their respective symmetry score. Most symmetry detection algorithms produce these attributes to draw symmetry lines. We tested our final algorithm on 200 images. The results of this, together with the source code, can be found on GitHub[1].

1.3 Types of Symmetry

Symmetries can be categorised into four different types, namely:

- Reflection - where the image appears mirrored alongside a straight axis. Figure 1a shows an example where the left half of the image is a mirrored version of the right half.
- Rotational - where elements are repeated around a centre point. Figure 1b shows an example where all the spokes of the wheel are centred around the middle of the ferris wheel.
- Translation - where elements are repeated while rotated in a different direction. Figure 1c displays an example where the metallic street stones are repeated in a different direction.
- Glide reflection - is a combination of reflection symmetry and translation symmetry. Figure 1d shows an image where the footprints are mirrored along a straight axis and are also translated in a forward direction.

Besides these four mentioned symmetry types, some algorithms focus on finding the medial axis (skeleton) of objects inside images. An example is shown in Figure 2.



(a) Reflection (b) Rotation (c) Translation (d) Glide-reflection

Figure 1: The four types of symmetry with the symmetry axis shown in yellow (source: [7])

[1] https://github.com/Koen-Git/ColorSymDetect

Figure 2: Medial axis symmetry with the axis drawn in yellow (source: [7])

1.4 Thesis Overview

In Section 2 we outline the related work. In Section 3 we describe the two algorithms on which we experimented and list their initial results. Furthermore, we describe possible improvements to these algorithms as well as describe and evaluate our machine learning approach to find rotational symmetries. In Section 4 we summarise our findings and results. In Section 5 we discuss shortcomings of our approach and outline future work.

This thesis is a bachelor project supervised by Bas van Stein and Anna Kononova and is performed at the Leiden Institute of Advanced Computer Science (LIACS).

2 Related Work

The first symmetry detection method was proposed by Atallah in 1984 [2, 17]. This algorithm focused on reflection symmetries, as shown in Figure 3. The main technique used has been called intensity-based detection [7] and has been proven by Cicconet et al. [6] to still be competitive in recent years [9]. However, this technique suffers most notably from requiring large computation time. This problem has increased with time as images have grown in resolution.

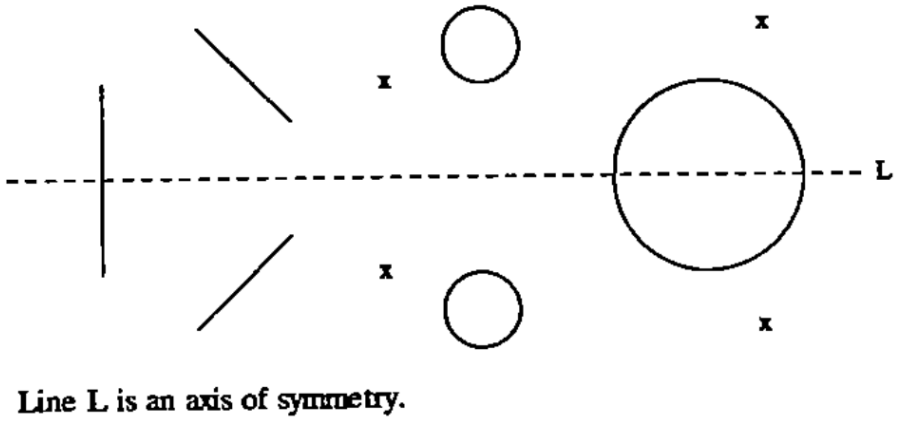


Figure 3: First symmetry detection algorithm (source: [2])

In more recent years, three different symmetry detection competitions were held in 2011, 2013 and 2017. We will now discuss each competitions findings and their key algorithms as the algorithms participating in these competitions were state of the art during their respective time.

Table 1: Details about the first symmetry detection competition (2011)

First symmetry detection competition [22]	
Year	2011
Organizer	CVPR
Categories	Reflection, rotation, translation
# participants	Reflection: 3, rotation: 2, translation: 3 (8 total)
Baseline algorithms	Loy and Eklundh [16] for reflection and rotation, [20] for translation.
Winners	Reflection: Baseline, rotation: Baseline, translation: unknown

First Symmetry Detection Competition
The first competition featured three different categories: reflection, rotation and translation. A total of 8 algorithms participated, with each category also featuring a baseline algorithm. The algorithms performed best on reflection symmetry, second-best on rotation symmetry and last on translation symmetry. For both the reflection and rotation category, all contestants were beaten by the baseline algorithm published by Loy and Eklundh in 2006 [16]. The results for translation symmetry can be disregarded as the participants did not conform to the evaluation standard and could not properly be evaluated.

Table 2: Details about the second symmetry detection competition (2013)

Second symmetry detection competition [14]	
Year	2013
Organizer	CVPR
Categories	Reflection, rotation, translation
# participants	Reflection: 3, rotation: 1, translation: 2 (6 total)
Baseline algorithms	[16] for reflection and rotation, [20] for translation symmetry.
Winners	Reflection: Baseline or [21] in some categories such as multiple symmetries, rotation: [16] in single centre detection and [12] for multiple centres, translation: [3].

Second Symmetry Detection Competition
The second competition, held in 2013, featured the same three categories. The baseline algorithm was again very competitive and outperformed most algorithms in reflection and rotation symmetries when only a single symmetry is considered. For multiple symmetries Patraucean [21] and Petrosino [12] won these categories respectively. For translation symmetry, Cai and Baciu [3] beat the baseline algorithm and the contestants, but it should be noted that this algorithm required manual input while the others did not. The baseline algorithm and the last contestant [20] could not be compared against each other.

Table 3: Details about the third symmetry detection competition (2017)

Third symmetry detection competition [9]	
Year	2017
organizer Organizer	ICCV
Categories	Reflection, rotation, translation, medial axis, 3-d data sets have also been added for these categories
# participants	11 participants
Baseline algorithms	reflection: [16] and [24], rotation: [16] and translation: [26].
Winners*	Reflection: Baseline [16] for multiple symmetries and [24] for single symmetries. rotation: unknown, translation: [26]
*The winners listed are on only on 2-d data as 3-d images are beyond the scope of this thesis	

Third Symmetry Detection Competition

The latest symmetry detection competition, held in 2017, featured the medial axis detection category for the first time. Loy and Eklundh [16] still acted as the baseline for reflection and rotational symmetries. A second baseline algorithm [24] for these categories was introduced and outperformed Loy and Ekhlundh on single reflection symmetries. Surprisingly Loy and Eklundh once again won in detecting multiple symmetries, with Elawady et al. [8] coming in second place. For translation symmetries, the scores of the challenger [18] and baseline [26] algorithms were still relatively low, with F-scores[2] of 0.19 and 0.20 respectively. In the new category, medial axis, two algorithms managed to beat the baseline algorithm. Liu et al. [15] came in first place with an F-score of 0.73. For comparison, humans achieved an F-score of 0.80 [9]. Funk and Liu [10] participated in the rotational symmetry category and the authors themselves claim better performance over Loy and Eklundh. However, Funk and Liu only focus on finding the rotational centres. In the results of the competition, no winner of the rotational symmetries category is discussed.

Figure 4: Example image of the deepflux algorithm [25]

[2]F-score (1) offers a single-value score for system performance comparison, by using the precision and recall [9]

Later Works

Since the last symmetry detection competition, Wang et al. published a new deep learning algorithm for finding medial axis symmetries [25]. In this paper, the authors claim equal or better performance compared to the winner of the latest competition [15]. While Deepflux is still a deep learning method, much like Liu's algorithm, it differentiates itself in its learning objective. While earlier deep learning-based methods trained their model as a binary classification problem, Deepflux focuses on learning a context 'flux' of the objects skeletons. This flux should result in more informative non-local cues, which lowers inaccuracy. An example of this flux is pictured in Figure 4.

In 2019 Elawady published his thesis [7] in which he states the evaluation method for reflection symmetry used in the competitions is unclear. Elawady explains how to evaluate symmetry axis to their corresponding ground truths. Just like the competitions, he uses the $max\{F1\}$ score (1) as a global quality measure for comparing algorithms.

$$max\{F1\} = max\{2\frac{PR \times RC}{PR + RC}\} \tag{1}$$

Furthermore, Elawady emphasizes that a pair of similar symmetry lines should not both be evaluated. Instead, only the higher ranked symmetry line (with a higher score) should be considered. Next, he states that multiple detected symmetry axes can be grouped to form a single ground truth. Both practices are visualised in Figure 5. It is unclear whether the symmetry competitions also adopted these methods, as they only list the precision and recall with their corresponding F-scores on a single test set.

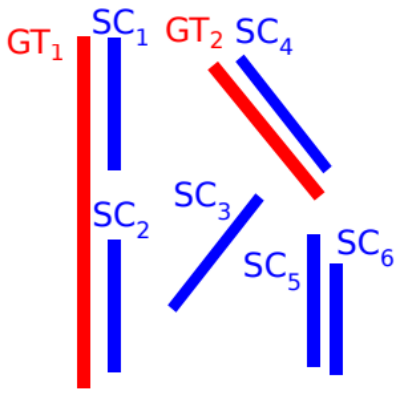


Figure 5: Example with two ground truths (red) and six detected symmetry lines (blue). The detected symmetry lines are ordered by their respective symmetry score produced by the algorithm. SC1 and SC2 are matched to GT1 (the difference in their slope and their centres are within some threshold) and SC4 is matched to GT2. Furthermore, SC6 can be disregarded as it is similar to the higher-ranked SC5. (source: [7])

Elawady has evaluated different versions of his algorithm on seven different symmetry data sets, four for single symmetry detection and three for multiple axis detection. The results are listed in Table 4. These results show that Elawady's algorithm has a slight edge in the last competition's multiple axis training data set (ICCV17m). The test set of this same competition shows Loy has the higher performance [9]. Nonetheless, when considering the other 6 data sets, we can see Elewady has a clear advantage in each of them over Loy and Eklundh.

Table 4: Elawady algorithm compared to Loy and Eklundh. and Cicconet et al. on seven different data sets. Each cell lists the amount of top-ranked detected symmetry axis that corresponds with a ground truth in the data set. Between brackets the $max\{F1\}$ scores are listed. (data source: [7])

Dataset (#images)	Loy [16]	Cic [5]	Elawady [8]
PSU (157)	81 (0.514)	90 (0.569)	**113 (0.724)**
AVA (253)	174 (0.690)	124 (0.493)	**182 (0.729)**
NY (176)	98 (0.528)	92 (0.526)	**135 (0.766)**
ICCV17 (100)	52 (0.507)	53 (0.536)	**70 (0.713)**
PSUm (142)	69 (0.292)	68 (0.159)	**75 (0.338)**
NYm (63)	32 (0.337)	36 (0.237)	**40 (0.411)**
ICCV17m (100)	54 (0.273)	39 (0.207)	**57 (0.285)**
Total (991)	560 (0.449)	502 (0.390)	**672 (0.567)**

Summary

Over the years, many papers [26, 10, 13] have proposed various algorithms to find symmetries in an automated way, yet it still poses to be a complex problem. According to all competitions, Loy and Eklundh [16] beat most other algorithms in various categories, even ten years after its publication. We also note that F-scores were still quite low during the last competition for multiple and single reflection symmetry detection (F=0.3 and F=0.52 respectively). Elawady has described more clearly how reflection symmetries should be evaluated and tested his algorithm on seven different data sets. His results show higher F-scores (0.73 for single reflection data sets and 0.34 for multiple reflection data sets). However, it is clear that there is still much room for improvement in this field.

For detecting medial axis symmetries, the F-scores are nearing those of humans [9]. This indicates that automated medial axis symmetry detection is close to producing more accurate results than humans would produce manually. Since the last symmetry detection competition, a more recent algorithm[25] is proposed, which should make this gap between automation and manual work even closer.

Detecting rotational symmetries has not received much attention in the last symmetry competition. Loy and Eklundh [16] won this category in the first and second symmetry competition and is considered the best baseline for this category. Funk and Liu [10] claim better performance but only focus on finding the rotational centres in their work.

Translation symmetry detection has proven to be the most challenging category. In the latest symmetry detection algorithm, the maximum achieved F-score was only 0.2.

3 Methodology

Based on our literature review, we conclude that for reflection symmetries, Elawady's WaveletSym algorithm [8] achieves the highest performance on the largest distribution of data sets while using a clear evaluation method. Furthermore, We found that WaveletSym processes images in under ten seconds on average. In Section 3.3, we compare the computation time of our work with Loy and Eklundh. We find that Loy and Eklundh's method requires four minutes on average to process an image. This time is comparable to our proposed work in which we recursively use WaveletSym

multiple times. For these reasons, we consider WaveletSym currently state of the art. In detecting medial axis symmetries, we consider Deepflux [25] the state-of-the-art algorithm, as the authors claim improved performance over the winner [15] in the third symmetry detection competition. To summarize, we will compare state-of-the-art algorithms WaveletSym and Deepflux on our data. WaveletSym uses an unsupervised method in order to find global reflection symmetries. Deepflux uses a deep learning approach to find the medial axis symmetries. For these reasons, it will be interesting to see which approach will work better on our data set and how we can best utilize both existing algorithms on our data set. For detecting rotational symmetries, we will not use an existing algorithm meant for this task. We instead propose a machine learning model in Section 3.3. This model uses reflection symmetry lines generated by WaveletSym. We will subsequently compare the performance of our model against Loy and Eklundh [16] in the same Section. Due to the current low performance of all translation symmetry detection algorithms [9], none of these methods are considered.

3.1 Deepflux

We initially ran the Deepflux algorithm with default settings and a provided model on a subset of images from our data set. We found that Deepflux with this provided model showed poor results when directly applied to the Ornamika data set, as shown in Figure 6. The skeletons of the important parts of the ornaments have not been identified, e.g. the bird (top left) or deer (bottom right) have no skeleton drawn on them.

Figure 6: Deepflux [25] with default settings from left to right: initial image — drawn skeleton — skeleton drawn on image

The provided models are trained on the SK-Large [23] and symPASCAL [11] data sets. These data sets mostly contain real-life images, as seen for example in Figure 4. Because our data set contains a different kind of images, we created a new model from our set of images to fully test Deepflux.

(a) original image (b) Generated skeleton (c) Skeleton drawn on image

Figure 7: Deepflux [25] with a new model trained on a small subset of the Ornamika [1] data set. These results show good performance for a small training set.

To train the new model, we started with a small set of 11 images and manually created ground truths for each image. Then, we trained the model using the same approach as described in the Deepflux paper [25], though with a different configuration of hardware and without using pycaffe's GPU mode. This should not impact any of the results, only the time it took to train the model. While the model is only trained on a small set of images, it would give us insights into whether or not creating a larger training set would be a worthwhile investment of time. Unfortunately, the results show that it would not.

With the newly trained model, some images show that there is potentially more to gain from increasing our training data size, as shown in Figure 7. We can see in the top images that the skeleton is quite accurately drawn on the important parts of the ornaments, as is the case in the bottom image. Even with a small training set, the shape of the bird is starting to form. Unfortunately, most of the images we tested on showed a different kind of performance which we can see in Figure 8 and 9. Deepflux either produces seemingly random scattered dots in these images or has drawn lines that did not form on the most important parts of the ornament.

Based on the resulting images (8, 9) and the viewed performance we just described, we conclude that enlarging the training data would very likely not be a worthwhile investment of time. Medial axis symmetry detection methods are, in their current state, probably not useful for our kind of data set. This may be because the images often do not contain any depth, and a change of colour may not always indicate a different object in the image. More research can be done in this field. Some ideas are shared in Section 5.

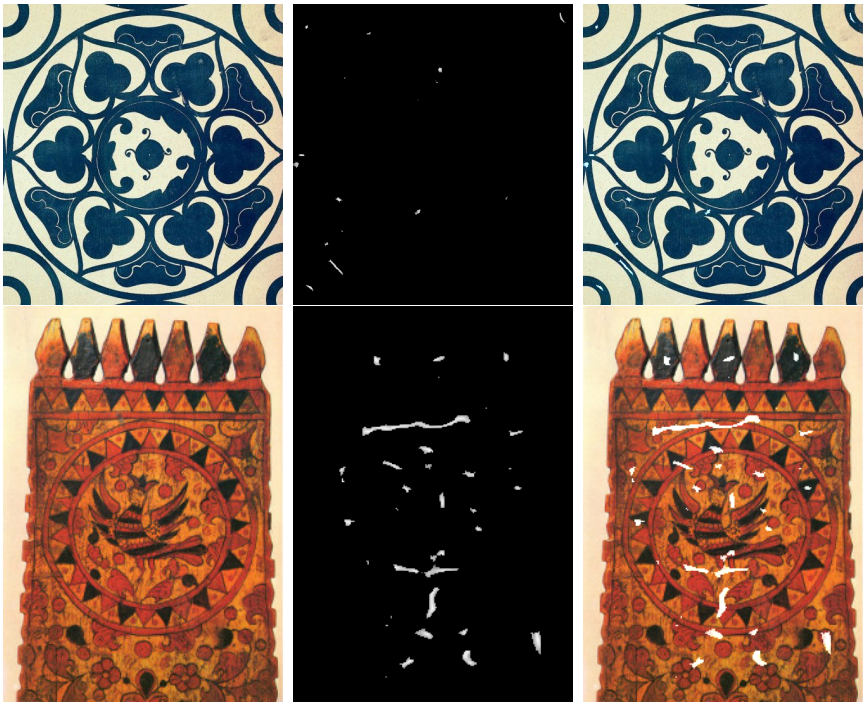


Figure 8: Deepflux [25] with a new model trained on a small subset of the Ornamika [1] data set. These images show the model is not providing accurate results.

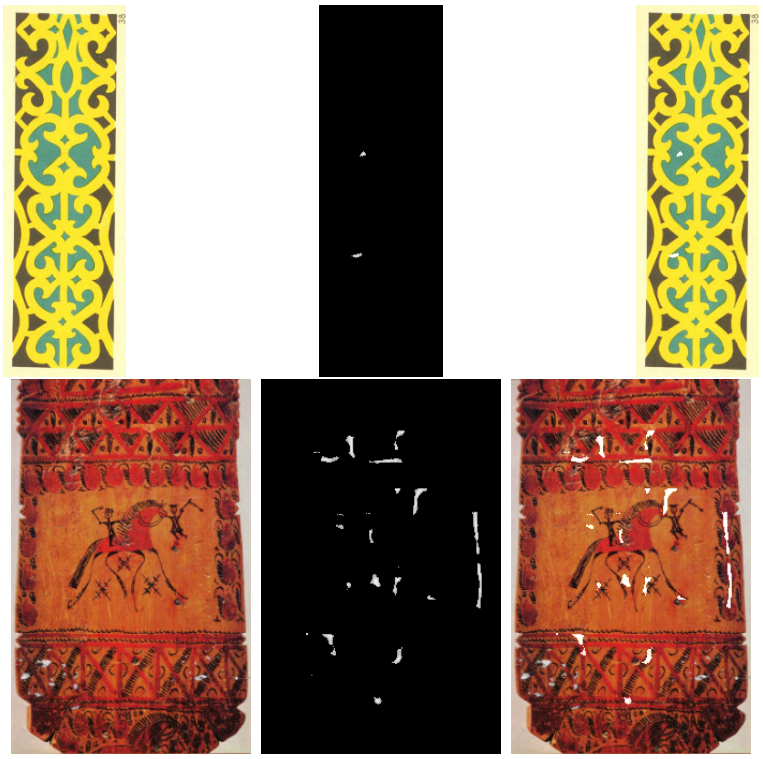


Figure 9: More examples of Deepflux [25] showing poor performance with our newly trained model, as is the case in Figure 8.

3.2 Wavelet-based Reflection Symmetry Detection

Elawady's [8] WaveletSym algorithm produces a multidimensional array of symmetry lines with their corresponding symmetry score. As shown in Figure 10, these lines are then drawn on the image with the legend of the scores attached.

Figure 10: Results as produced by [7] without any alterations. The top 5 discovered symmetries are drawn on the image. The numbers in the legend represent each lines symmetry score, how strong of a symmetry is present.

Initial results

WaveletSym gives us pretty satisfying results, as is shown in Figure 10. With default settings, the top 5 symmetries are drawn on the image. We can see WaveletSym often finds the main symmetry axis, with secondary symmetries being found relatively accurate as well. We should note that in a

data set mainly containing art pieces like the Ornamika [1] data set, the main symmetry axis is often the least interesting. This is because in a vast majority of the images a reflection symmetry exists in the centre of the image.

When inspecting the resulting images in Figure 10 we observe a few shortcomings of the WaveletSym algorithm in its current state.

1. As is described in the paper [8], the algorithm only deals with global reflection symmetries, while we are also interested in more localised symmetries. Especially when considering that the main global symmetry axis is often the least interesting in our data set.

2. This algorithm only deals with reflection symmetries, but we are interested in other types as well, such as rotational symmetries.

3. All symmetry lines with a score over 0 are returned by the algorithm, with no way to filter on accurate symmetry lines. Yet, the scores themselves are not a very meaningful metric when comparing scores from one image to another.

4. The algorithm produces many symmetry lines that are very similar to each other.

5. The symmetry detection itself could be improved.

Improvements
Despite these shortcomings, WaveletSym is still a promising algorithm, and some alterations will mitigate these described problems. In the next paragraph, we will go through these problems one by one and attempt to mitigate them as much as possible.

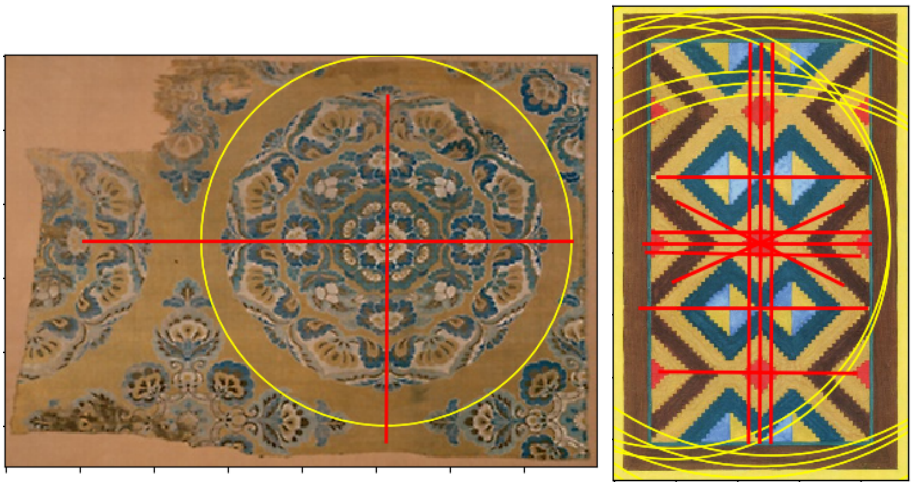
1. **Localised symmetries:** We will cut the images at certain points to find localised symmetries in these smaller parts. Then, the resulting symmetry lines can be plotted on the original image, revealing the more localised symmetries. For determining where to cut the image, we could cut at as many places as possible. However, this results in undesired symmetry lines and most notably increases computation time. Instead, we settled on recursively cutting up the image at the top three symmetry lines. This created six new smaller images which can be processed in the same manner. This resulted in a good balance between computation time and retrieving accurate symmetries. The pseudo-code for this recursive function is displayed in Algorithm 1.

2. **Rotational symmetry:** First, we observed that when two lines cross and are perpendicular, there is likely a rotational symmetry around the intersection as shown in Figure 11a. The rotational symmetries as drawn with this naive approach lead to some unwanted results, as shown in Figure 11b. These results are partly caused by our earlier described problem of similar symmetry lines being drawn close to each other; this also causes multiple rotational symmetries to be drawn. The other problem is that there are not necessarily rotational symmetries at the intersections between two perpendicular lines. Requiring the two perpendicular lines to have similar symmetry scores will improve performance. Besides these simple rules, we also contribute a machine learning approach for detecting rotational symmetries with symmetry line pairs as an input. We will elaborate more on this in Section 3.3.

3. **Filtering:** To determine if symmetries should be drawn on the image, we can use the normalised score for each image or sub-image. First, we checked if the top symmetry found reaches our parameter symmetry threshold, and then we check if the other detected symmetry scores lie within a second threshold compared to the first one. Symmetries that do not reach these thresholds can be removed.

4. **Similar symmetries:** Since WaveletSym is producing a lot of similar symmetry lines, we can loop over each line and compare them to each other. If we find lines with a similar slope and centre, we can discard the line with a lower symmetry score. The same procedure can be performed on rotational symmetries, i.e. circles drawn with a similar radius and centre.

5. **Improvements:** In order to improve the detection, we could highlight the important details of each ornament. We attempted this by using Deepflux [25] to draw skeletons on the ornaments. However, as we described in Section 3.1, this approach yielded no good results.



(a) Naive approach for rotational symmetries in a perfect scenario

(b) Naive approach for rotational symmetries in a bad scenario

Figure 11: Rotational symmetries drawn with with a naive approach. The red lines represent the discovered reflection symmetries. Any perpendicular lines will draw a yellow rotational symmetry in their intersection.

Results after improvements

The pseudo-code for the algorithm with our improvements is displayed in Algorithm 1. Figure 12 lists images with all our proposed adjustments. The rotational symmetries are detected in these images using the described naive approach, not using machine learning. For this set, we have set our first parameter, 'globalSymmetryThreshold', to 0.20. This parameter dictates whether the top symmetry of each image or sub-image should be kept or not. Any lines with scores below this parameter are removed. Lines generated as part of the same recursive loop are removed as well. We have set our second parameter, 'normalizedThreshold', to 0.70. This parameter dictates which other symmetry lines are kept. Lines with scores relative to the top symmetry score which are lower

than this threshold, are removed. The third parameter, 'CircleSymThreshold', was set to 0.75; this implies the maximum difference in symmetry scores between two lines can at most be 25% for them to create a rotational symmetry.
The resulting images show that with these adjustments, WaveletSym is capable of finding localised symmetries and rotational symmetries. However, some rotational symmetries are still not found, and some smaller ones are found inaccurately. Additionally, some localised symmetries which are discovered are inaccurate.

Figure 12: Resulting images of WaveletSym with our alterations outlined in Section 3.2. The blue lines are the discovered global symmetries. Other colours represent different depths of the recursive function and, in turn, represent discovered localised symmetries. The colours of the different depths are mapped in the colour map to the right. Colours presented lower on the map represent more localised symmetries.

Algorithm 1 WaveletSym with our post-processing improvements

```
function recursiveSymmetries(img, symmetries, minsize, rcAmount)
    if img.height < minsize[0] or img.width < minsize[1] then
        return
    end if
    newSymmetries = WaveletSym(img) [8]    ▷ newSymmetries is ordered by symmetry score
    for i in range(0, rcAmount) do
        (im1, im2) = cut image along newSymmetries[i]
        symmetries + = recursiveSymmetries(im1, symmetries, minsize, rcAmount)
        symmetries + = recursiveSymmetries(im2, symmetries, minsize, rcAmount)
    end for
    symmetries + = newSymmetries
    return symmetries
end function

function rotationalSymmetries(symmetries, img, circleSymThreshold)
    rotations = []
    for sym in symmetries do
        for subsym in symmetries do
            if abs(sym.score - subsym.score) < circleSymThreshold then
                if isPerpendicular(sym, subsym) then
                    intersect = calcIntersection(sym, subsym)
                    if intersect != None then
                        if checkDistance(intersect, sym, subsym) then
                            radius = minDistance(intersect, sym, subsym)
                            rotations.append([intersect, radius])
                        end if
                    end if
                end if
            end if
        end for
    end for
end function

for img in folder do
    symmetries = recursiveSymmetries(img, [], (img.height/5, img.width/5), rcAmount)
    symmetries = removeSymmetries(symmetries, symThreshold, normThreshold)
                        ▷ Remove symmetries with low symmetry scores (under threshold)
    rotations = rotationalSymmetries(symmetries, img, circleSymThreshold)
    symmetries = removeSimilarReflections(symmetries, similarityThreshold)
                                                ▷ Remove lines with similar slope and centre
    rotations = removeSimilarRotations(rotations, circleSimilarityThreshold)
                                        ▷ Remove rotations with similar radii and centre points
    PlotSymmetries(symmetries, rotations)
                                        ▷ Plot each reflection symmetry and rotational symmetry
end for
```

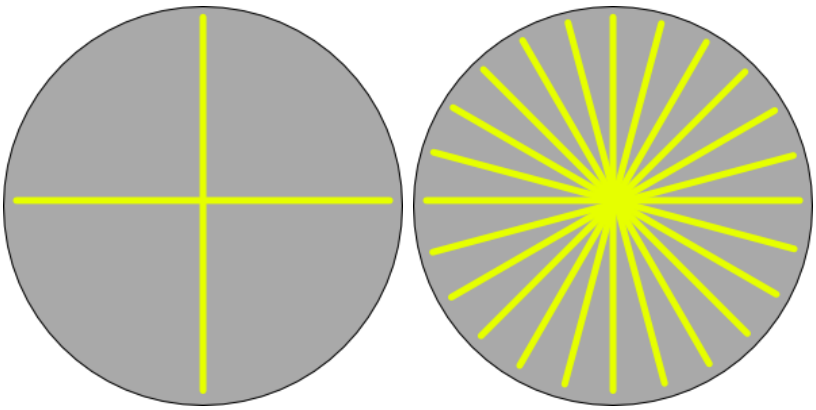


Figure 13: The image on the left represents two symmetry lines with a rotational symmetry in their intersection. In order to oversample, we rotate these lines around their centre point, as shown in the right image. In the actual oversampling, this is done 720 times, each rotated 0.5 degrees.

3.3 Machine Learning for Rotational Symmetries

Discovering rotational symmetries as described in Section 3.2 amounts to reasonable results. In order to improve these results we, propose to use a machine learning model to learn when reflection symmetry lines indicate a rotational symmetry at their intersection. For this classification task, we first had to label symmetry line combinations manually. We labelled lines that intersected and had a rotational symmetry at the intersection as true. Lines were also required not to be parallel or close to parallel. Other line pairs are labelled as false. In order to balance the data set, we also rotated every true case by 0.5 degrees 720 times. This process is visualised in Figure 13. We compared various machine learning algorithms trained on training data and validated on a different set of validation data. These sets contain 153.785 and 517.213 samples respectively. The results for each algorithm are shown in Table 5.

Table 5: Accuracy scores off various algorithms on our validation data. While SVC_auto achieves the highest accuracy, the ROC curve (Figure 14) shows that Random Forest and LinearSVC perform the best.

Algorithm	**Parameters***	**Accuracy**	**AUC**
Log_Regression	max_iter=10000	0.82181	0.5
Random Forest		0.85381	**0.94**
SVC		0.19588	0.51
SVC_alt	gamma="auto"	**0.88888**	0.76
KNN		0.86131	0.71
Decision Tree		0.87025	0.67
SGDClassifier		0.46510	0.58
LinearSVC	max_iter=5000	0.88024	**0.94**
*All non listed parameters are default parameters. Using sklearn 0.24.2 (python 3.7.10). Each algorithm, except for KNN, was also set with "random_state=44" and "class_weight='balanced'".			

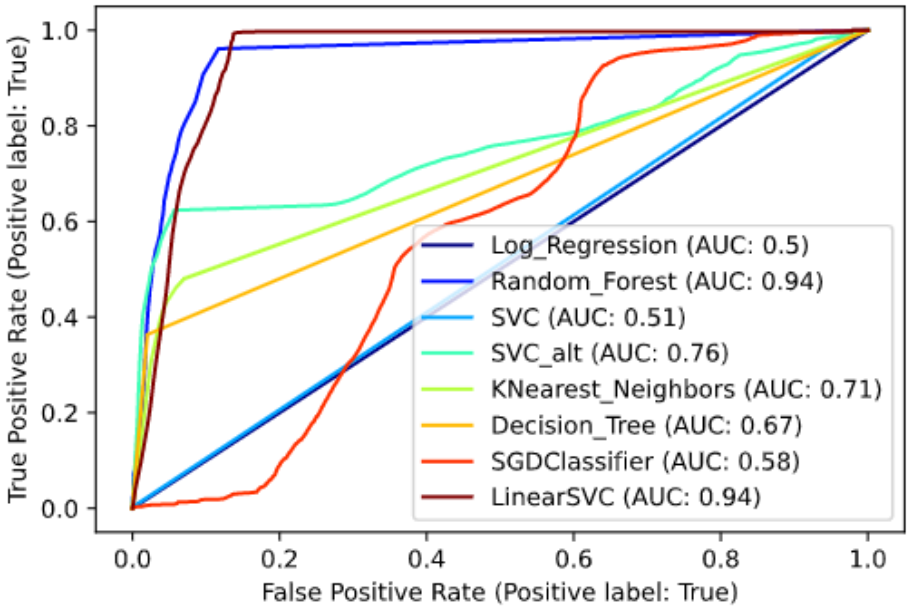


Figure 14: ROC Curve of the scores in Table 5. Random Forest and LinearSVC come out on top.

Table 6: KNN and SVC accuracy scores and AUC scores with different hyper parameters. The algorithms are trained on our training data and validated on a separate validation set. The top scores are listed in bold text.

Algorithm	**Parameters**	**accuracy**	**AUC**
KNN		0.86131	0.71
KNN	n_neighbors=1	0.84903	0.65
KNN	n_neighbors=3	0.85738	0.69
KNN	n_neighbors=8	**0.86407**	**0.72**
KNN	weights="distance"	0.86069	0.71
KNN	weights="distance", n_neighbors=3	0.85513	0.69
KNN	weights="distance", n_neighbors=8	0.86213	**0.72**
SVC	gamma='auto'	0.88888	0.75
SVC	gamma='auto', kernel='poly'	0.17819	0.5
SVC	gamma='auto', kernel='sigmoid'	0.31574	0.42
*All non listed parameters are default parameters. Using sklearn 0.24.2 (python 3.7.10). All SVC configurations were also using "class_weight='balanced'" and "random_state=44".			

Table 7: Accuracy scores and AUC scores of Random Forest with different hyper parameter configurations trained on training data and validated on a separate validation set. The highest scores are listed in bold text.

id	**Parameters**	**accuracy**	**AUC**
0		0.85381	0.94
1	n_estimators=50	0.85419	0.91
2	n_estimators=250	0.85496	0.95
3	n_estimators=500	0.85651	0.95
4	criterion="entropy"	0.87617	0.96
5	criterion="entropy", n_estimators=250	0.87781	0.96
6	max_depth=5	0.89620	0.96
7	max_depth=20	0.88323	0.95
8	min_samples_split=4	0.85769	0.94
9	min_samples_leaf=3	0.88524	0.95
10	max_features="log2"	0.85381	0.94
11	max_leaf_nodes=10	0.87676	0.96
12	max_depth=5, criterion="entropy"	0.89622	0.96
13	max_depth=10, criterion="entropy"	**0.90819**	0.96
14	max_depth=20, criterion="entropy"	0.88375	0.96
15	min_samples_leaf=3, max_depth=5	0.89620	0.96
16	min_samples_leaf=3, max_depth=20	0.89225	0.95
17	min_samples_leaf=3, criterion="entropy"	0.89420	0.96
18	min_samples_leaf=3, criterion="entropy", max_depth=5	0.89622	0.96
19	min_samples_leaf=3, criterion="entropy", max_depth=20	0.89559	**0.97**
*All non listed parameters are default parameters. Using sklearn 0.24.2 (python 3.7.10). Each algorithm was also set with "random_state=44" and "class_weight='balanced'".			

The results show that SVC, with its 'gamma' hyperparameter set to "auto" achieved the highest accuracy. However, in the ROC curve, it is clear SVC is scoring poorly with an Area Under Curve (AUC) of only 0.76. Random Forest and LinearSVC achieved better performance with an AUC of 0.94.

We re-evaluated these three top-performing algorithms with different hyperparameter configurations. Together with these algorithms, we also evaluated KNN, since it was the only other algorithm achieving an AUC > 0.70. Both SVC and KNN showed no noteworthy improvements in their AUC scores, as shown in Table 6. Different hyperparameters showed no improvements in LinearSVC either but did show some small improvements in Random Forest. The results are listed in Table 7 and 8. We evaluated the top-performing Random Forest and LinearSVC configurations one more time on a completely different test set, containing 263.569 samples. The final results of this test are found in Table 9, and the ROC curve is found in Figure 15. We conclude that Random Forest with "max_depth=10" and "criterion='entropy'" performs the best on various sets of our data.

Table 8: LinearSVC Algorithm accuracy scores and AUC scores with different hyper parameter configurations. The algorithms are trained on our training data and validated on a separate validation set. The top scores are listed in bold text.

id	**Parameters**	**accuracy**	**AUC**
0	max_iter=20000	**0.88566**	0.94
1	max_iter=20000, loss="hinge"	0.82690	0.91
2	max_iter=20000, C=0.1	0.88107	0.94
3	max_iter=20000, dual=False	0.45298	0.57
4	max_iter=20000, dual=False, penalty="l1"	0.86240	**0.95**
5	max_iter=20000, dual=False, C=10	0.45298	0.57
*All non listed parameters are default parameters. Using sklearn 0.24.2 (python 3.7.10). Each algorithm was also set with "random_state=44" and "class_weight='balanced'".			

Table 9: Accuracy scores of the best performing algorithms with their hyper parameters tuned. These tests are performed on a separate test set, which is entirely different from the earlier validation set.

algorithm	**Parameters**	**accuracy**	**AUC**
Random Forest	max_depth_10, criterion="entropy"	**0.97348**	**0.996**
Random Forest	min_samples_leaf=3, criterion="entropy", max_depth=20	0.87452	0.994
LinearSVC	max_iter=20000	0.92990	0.969
LinearSVC	max_iter=20000, dual=False, penalty="l1"	0.92191	0.973
*All non listed parameters are default parameters. Using sklearn 0.24.2 (python 3.7.10). Each algorithm was also set with "random_state=44" and "class_weight='balanced'".			

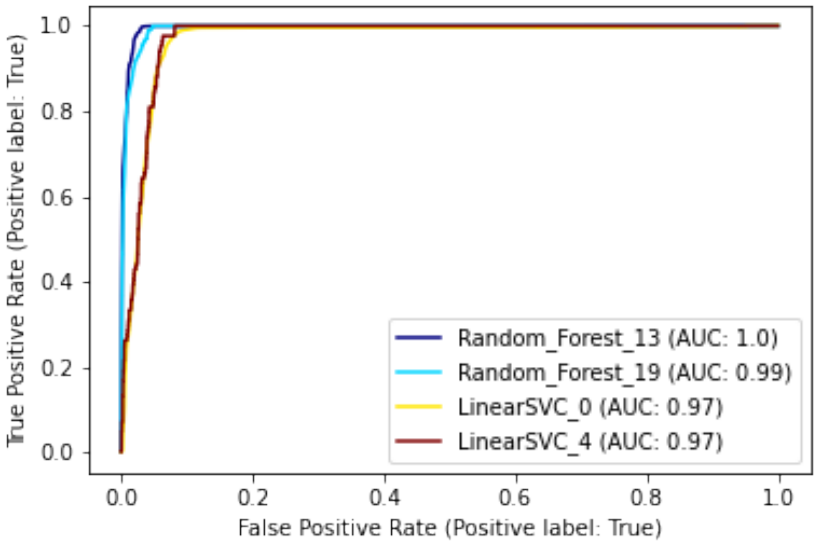


Figure 15: ROC curves of the different hyper parameter configurations listed in Table 9

Instead of our earlier naive approach to detect rotational symmetries as described in Section 3.2, we can now use our newly trained Random Forest model on each line pair that is found by the algorithm. Any true case will have the centre of its symmetry at the intersection of the reflection symmetry line pair.

We found that our model performs well in identifying rotational symmetries based on two given reflection symmetry lines. However, this does mean that our detection method is still completely reliant on the provided reflection symmetry lines. Due to this, we will now remove fewer symmetry lines as we did in the previous Section (3.2) since we can use the model to determine if these lines help detect rotational symmetries. After the predictions are made, we will remove the low scoring symmetries as usual. This provides us with a better detection rate for rotational symmetries at the cost of computation time. The final-pseudo code is listed in Algorithm 2.

Figures 16 and 17 display various images with the original algorithm on the left, our improvements in Section 3.2 in the middle, and on the right our improvements with the machine learning model for rotational symmetries. We can see the machine learning model has improved the detection rate of especially the global rotational symmetries. In addition, some more localised rotational symmetries are now captured as well. The results do, however, show an increased amount of false positives.

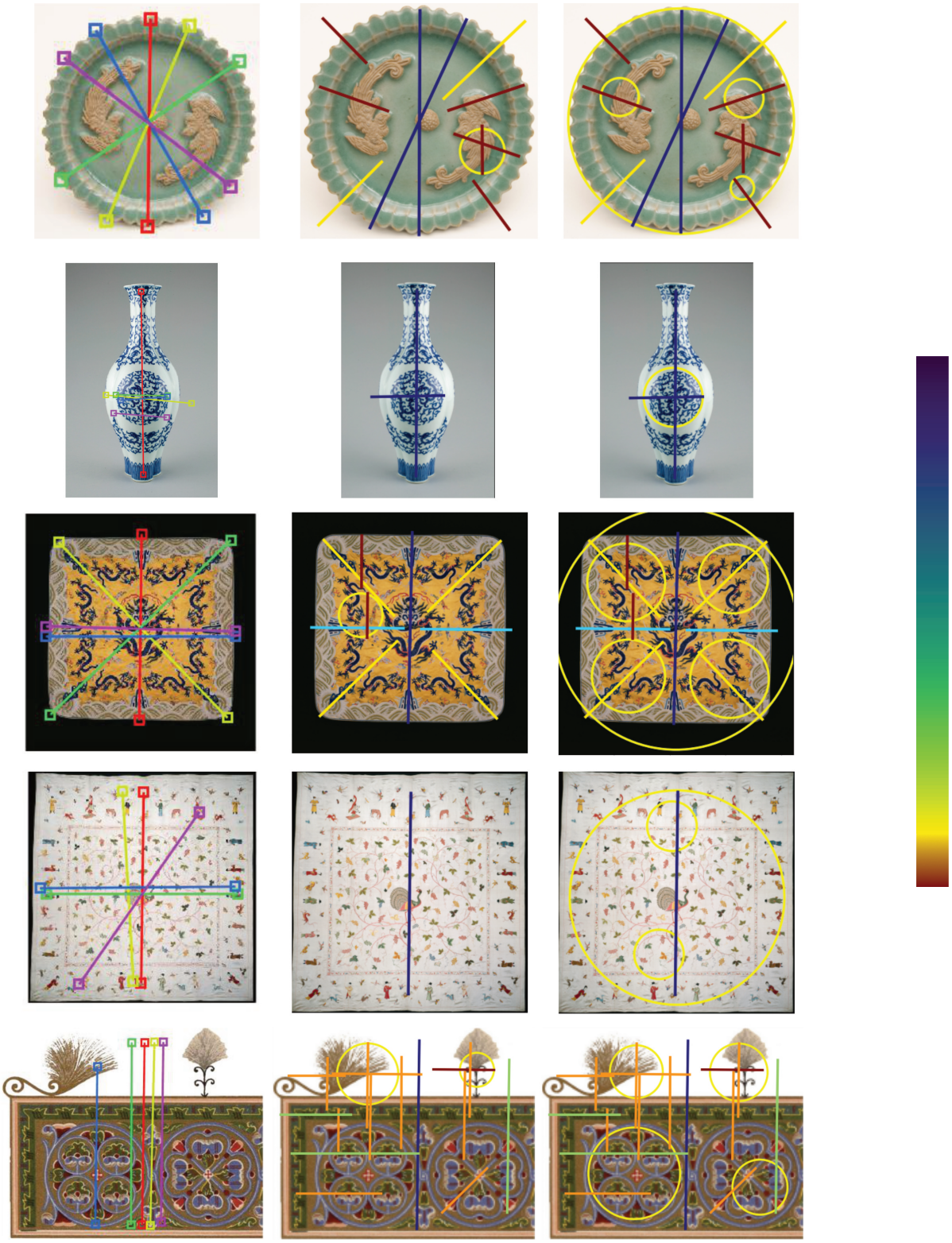


Figure 16: Original WaveletSym on the left, our work in Section 3.2 in the middle, and our machine learning model approach on the right. Rotational symmetries are drawn in yellow. The colour map on the right indicates different levels of localised reflection symmetries. Our machine learning model on the right accurately detects each global rotational symmetry and even some localised symmetries in the 2nd and 5th images. Our rule-based approach found no accurate rotational symmetries.

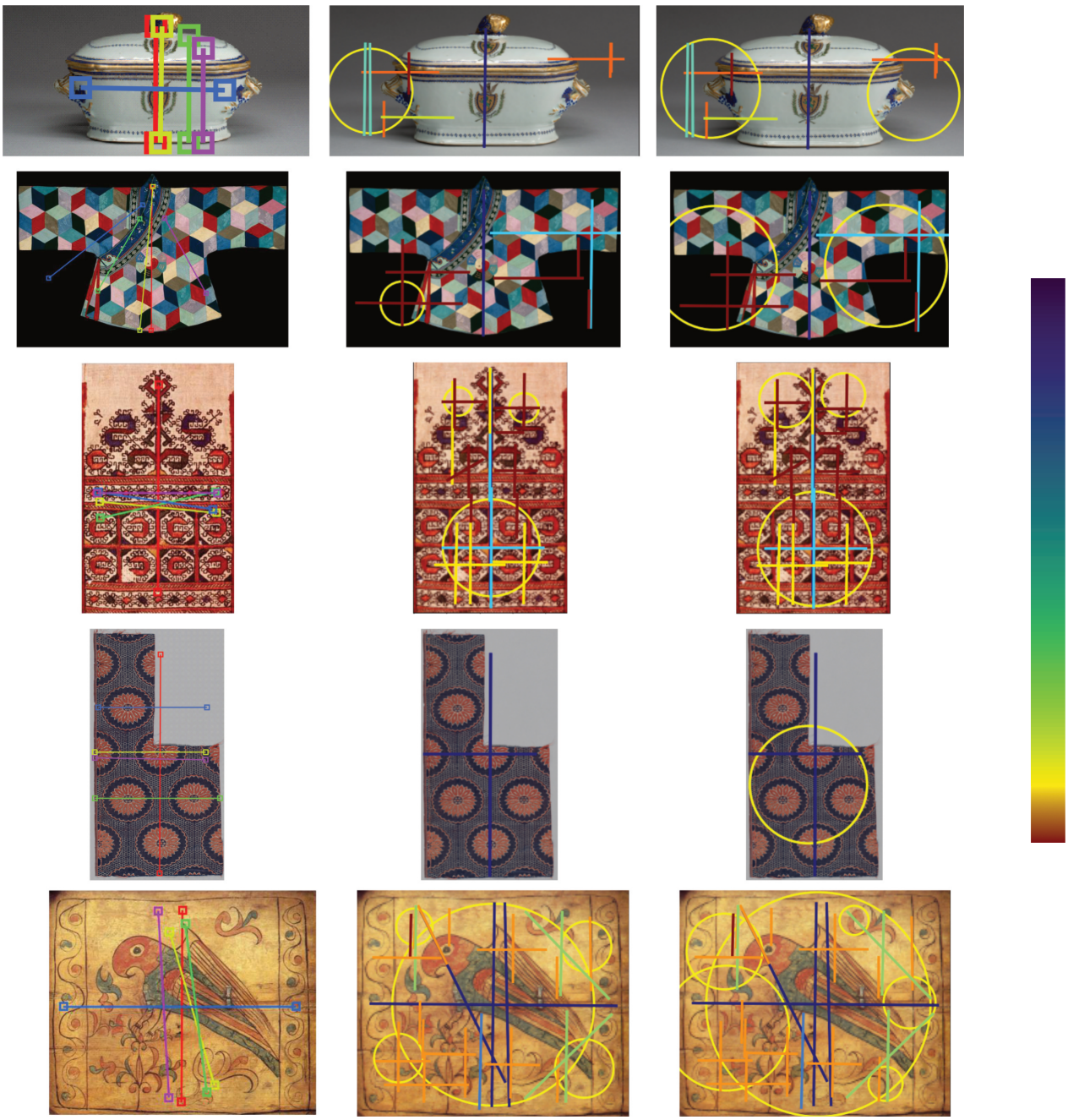


Figure 17: Original WaveletSym on the left, our work in Section 3.2 in the middle, and our machine learning model approach on the right. Rotational symmetries are drawn in yellow. The colour map on the right indicates different levels of localised reflection symmetries. These images show fail cases of our work. Both our rule-based method and machine learning method incorrectly identified rotational symmetries. Our machine learning model detects more false positives in the fourth image.

Algorithm 2 WaveletSym with our post-processing improvements and Machine learning model

```
function rotationalSymmetriesML(symmetries, img, model)
    rotations = []
    for sym in symmetries do
        for subsym in symmetries do
            intersect = calcIntersection(sym, subsym)
            if intersect != None then
                data = preprocess(sym, subsym)
                if model.predict(data) then
                    radius = minDistance(intersect, sym, subsym)
                    rotations.append([intersect, radius])
                end if
            end if
        end for
    end for
end function

for img in folder do
    symmetries = recursiveSymmetries(img, [], (img.height/5, img.width/5), rcAmount)  ▷ (1)
    symmetries = removeSymmetries(symmetries, lowSymThreshold, lowNormThreshold)
    rotations = rotationalSymmetriesML(symmetries, img, RF_model)
    symmetries = removeSymmetries(symmetries, symThreshold, normThreshold)
    symmetries = removeSimilarReflections(symmetries, similarityThreshold)
    rotations = removeSimilarRotations(rotations, circleSimilarityThreshold)
    PlotSymmetries(symmetries, rotations)
end for
```

Comparison with Loy and Eklundh

In Section 2 Loy and Eklundh [16] is described as a good benchmark for rotational symmetry detection methods, as it has won two competitions. Figure 18 and 19 show images of our proposed machine learning model against Loy and Eklundh's method. Our method shows superior performance when it comes to detecting rotational symmetries in images that are completely defined as a rotational symmetry (first, third and fifth image in Figure 18). Rotational symmetries have, by definition, an infinite amount of smaller rotational symmetries with the same centre. Our model will accurately draw the largest encompassing rotational symmetries in images defined entirely by a rotational symmetry. Loy's method does not always find the largest encompassing symmetry or detects a greater symmetry strength in the wrong centre. Furthermore, in all of our tested images, Loy's method found rotational symmetries in each of them. Our method will more often not produce false positives when there are no rotational symmetries in an image. Examples of this are shown in the first, second and third image in Figure 19. The last two images in Figure 19 show greater performance for Loy and Eklundh. Both methods struggle with detecting multiple rotational symmetries. However, Loy's method has a slight edge in this category. Most of the secondary rotational symmetries detected by our method are not accurate.

Since symmetries should often be detected quickly, or in our case, on a large number of images, the

speed of each algorithm is an important factor. Table 10 lists the average and median time taken by our work and Loy and Eklundh's method to detect rotational and reflection symmetries in a set of 10 images. As is clear from the low median, most rotational symmetries are detected very quickly with our method. However, some outliers do significantly raise the average time taken. It should be noted that our rotational symmetry detection method requires reflection symmetries as an input. This means the reflection symmetries need to be computed first. On the other hand, Loy and Eklundh's method is able to detect rotational symmetries independently. Furthermore, during testing, we found that Loy and Eklundh's algorithm required a lot of computational power. Running two instances simultaneously caused our computer to slow down severely during some processing stages. In contrast, our method was efficiently running on 6 or 7 instances, only being constraint by our RAM[3]. This means that, depending on the computer configuration, it will be possible to decrease the average detection time by several factors. This greatly benefits cases where lots of images need to be processed, such as with our case for labelling the Ornamika database.

Table 10: Average and median computation times of Loy [16] and our work in both rotation and reflection symmetry detection on a set of 10 images. Our work requires reflection symmetries to be detected before being able to detect rotational symmetries.

	Our work (reflection)	Loy (reflection)	Our work (rotation)	Loy (rotation)
Average time (s)	171	239	256	95
Median time (s)	109	103	3	38

[3]Our computer configuration contained an AMD Ryzen 3600, with 16GB of RAM.

Figure 18: Our method (right) compared against Loy and Eklundh [16] (middle). The left column contains the ground truths of the images drawn in red. These images show superior or equal performance of our model over Loy and Eklundh in detecting rotational symmetries.

Figure 19: Our method (right) compared against Loy and Eklundh [16] (middle). The left column contains the ground truths of the images drawn in red. The first three images contain no rotational symmetries. Our method is better able to detect this. The bottom three images show superior performance of Loy and Eklundh over our model.

4 Conclusion

In this thesis, we evaluated two state-of-the-art symmetry detection algorithms on the images of the Ornamika data set [1] and made improvements to one of them.

We found that the Deepflux [25] algorithm for detecting medial axis symmetries did not perform well on our data. We tried creating a new model trained on a small set of our data to see if it improved the results. Unfortunately, this has not paid off. We conclude that current medial axis symmetry detection algorithms appear to perform better on real-life photographic images.

The second algorithm [8] we evaluated did give us quite satisfactory results on our data set, even without alterations. We made alterations to automate the identification of accurately detected symmetries and removing others. We created a recursive function to apply the algorithm on smaller parts of the image and, in turn, detect more localised symmetries. We also proposed a set of rules with which the algorithm was able to identify rotational symmetries and draw them on the images. Lastly, we proposed a machine learning model based on the Random Forest classifier to replace our set of rules for detecting rotational symmetries. We found that with our model, rotational symmetries are detected with an accuracy of 90% on our validation set and 97% on our final test set. As such, we recommend using the machine learning approach, as it is better able to detect rotational symmetries. We can conclude that, with our approach, it is possible to detect rotational symmetries with reflection symmetry detection methods. Finally, we compared our method of detecting rotational symmetries against Loy and Eklundh's method and found our work to produce more accurate results when a single rotational symmetry is considered.

5 Future Perspectives

Parallelization: Using multi threaded-processors would greatly benefit the computation time of this algorithm since multiple images could be processed simultaneously. We, unfortunately, did not get around to implementing this feature. However, in the current state, multiple instances can easily be created manually.

Medial axis symmetry detection: We concluded that medial axis symmetry detection in its current state is not suited for our kind of data set. We did, however, never entirely create a model with sufficiently large enough training data. As such, we cannot definitively say whether this approach leads to good or bad results. Furthermore, medial axis symmetry detection appeared to show good results when focusing on animal-like shapes in our data set. As such, medial axis symmetry detection could yield better results on similar data sets containing more animals as ornaments.

Increasing accuracy of localised symmetries: We found, through observations, lower-resolution images (or smaller parts of images) to produce higher symmetry scores in general with waveletSym. This could imply that a fixed threshold for identifying accurate symmetries is not the right approach. Instead, a higher threshold on each smaller part of images could improve the accuracy.

Rotational symmetries: Because our method relies on reflection symmetry detection methods, its performance should, in theory, increase as reflection symmetry detection methods improve. As such, as reflection detection improves, our method can be re-evaluated.
Furthermore, while Funk and Liu [10] only focuses on finding the centres of rotational symmetries, it could still be a useful algorithm, and we have not evaluated it. Our method uses the minimum distance between the centre and any reflection line endpoint to find the radii of rotational symmetries. It may be possible to use Liu's algorithm in conjunction with reflection symmetries to determine the sizes of the rotational symmetries in the same way.

References

[1] Ornamika Database. https://ornamika.com/. Accessed: 2021-06-18.

[2] Mikhail J Atallah. On Symmetry Detection. In *Department of Computer Science Technical Reports (paper 396), Purdue University*, 1984.

[3] Yunliang Cai and George Baciu. Translation Symmetry Detection: a Repetitive Pattern Analysis Approach. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 223–228, 2013.

[4] Minsu Cho and Kyoung Mu Lee. Bilateral Symmetry Detection via Symmetry-Growing. In *Proceedings of the British Machine Vision Conference*, pages 4.1–4.11. BMVA Press, 2009. doi:10.5244/C.23.4.

[5] Marcelo Cicconet, Davi Geiger, Kristin C Gunsalus, and Michael Werman. Mirror symmetry histograms for capturing geometric properties in images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2981–2986, 2014.

[6] Marcelo Cicconet, David GC Hildebrand, and Hunter Elliott. Finding Mirror Symmetry via Registration and Optimal Symmetric Pairwise Assignment of Curves. In *Proceedings, ICCV Workshop on Detecting Symmetry in the Wild*, pages 1749–1758, 2017.

[7] Mohamed Elawady. *Reflection Symmetry Detection in Images: Application to Photography Analysis*. PhD thesis, Université de Lyon, 2019.

[8] Mohamed Elawady, Christophe Ducottet, Olivier Alata, Cecile Barat, and Philippe Colantoni. Wavelet-based Reflection Symmetry Detection via Textural and Color Histograms. In *Proceedings, ICCV Workshop on Detecting Symmetry in the Wild*, volume 3, page 7, 2017.

[9] Christopher Funk, Seungkyu Lee, Martin R. Oswald, Stavros Tsogkas, Wei Shen, Andrea Cohen, Sven Dickinson, and Yanxi Liu. 2017 iccv challenge: Detecting Symmetry in the Wild. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 1692–1701, 2017.

[10] Christopher Funk and Yanxi Liu. Beyond Planar Symmetry: Modeling Human Perception of Reflection and Rotation Symmetries in the Wild. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 793–803, 2017.

[11] Wei Ke, Jie Chen, Jianbin Jiao, Guoying Zhao, and Qixiang Ye. SRN: Side-output Residual Network for Object Symmetry Detection in the Wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1068–1076, 2017.

[12] Shripad Kondra, Alfredo Petrosino, and Sara Iodice. Multi-scale Kernel Operators for Reflection and Rotation Symmetry: Further Achievements. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 217–222, 2013.

[13] Chang Liu, Wei Ke, Jianbin Jiao, and Qixiang Ye. Rsrn: Rich Side-output Residual Network for Medial Axis Detection. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1739–1743, 2017.

[14] Jingchen Liu, George Slota, Gang Zheng, Zhaohui Wu, Minwoo Park, Seungkyu Lee, Ingmar Rauschert, and Yanxi Liu. Symmetry Detection from Realworld Images Competition 2013: Summary and Results. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 200–205, 2013.

[15] Xiaolong Liu, Pengyuan Lyu, Xiang Bai, and Ming-Ming Cheng. Fusing Image and Segmentation Cues for Skeleton Extraction in the Wild. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1744–1748, 2017.

[16] Gareth Loy and Jan-Olof Eklundh. Detecting Symmetry and Symmetric Constellations of Features. In *European Conference on Computer Vision*, pages 508–521. Springer, 2006.

[17] Joseph Manning and Mikhail J Atallah. Fast Detection and Display of Symmetry in Trees. In *Department of Computer Science Technical Reports (paper 481), Purdue University*, 1985.

[18] Eckart Michaelsen and Michael Arens. Hierarchical Grouping using Gestalt Assessments. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1702–1709, 2017.

[19] Yansheng Ming, Hongdong Li, and Xuming He. Symmetry Detection via Contour Grouping. In *2013 IEEE International Conference on Image Processing*, pages 4259–4263. IEEE, 2013.

[20] Minwoo Park, Kyle Brocklehurst, Robert T Collins, and Yanxi Liu. Deformed Lattice Detection in Real-world Images using Mean-shift Belief Propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(10):1804–1816, 2009.

[21] Viorica Patraucean, Rafael Grompone von Gioi, and Maks Ovsjanikov. Detection of Mirror-symmetric Image Patches. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 211–216, 2013.

[22] Ingmar Rauschert, Kyle Brocklehurst, Somesh Kashyap, Jingchen Liu, and Yanxi Liu. First Symmetry Detection Competition: Summary and Results. *The Pennsylvania State University, PA, Tech. Rep. CSE11–012*, 2011.

[23] Wei Shen, Kai Zhao, Yuan Jiang, Yan Wang, Xiang Bai, and Alan Yuille. Deepskeleton: Learning Multi-task Scale-associated Deep Side Outputs for Object Skeleton Extraction in Natural Images. *IEEE Transactions on Image Processing*, 26(11):5298–5311, 2017.

[24] Tom Sie Ho Lee, Sanja Fidler, and Sven Dickinson. Detecting Curved Symmetric Parts using a Deformable Disc Model. In *Proceedings of the IEEE international conference on computer vision*, pages 1753–1760, 2013.

[25] Yukang Wang, Yongchao Xu, Stavros Tsogkas, Xiang Bai, Sven Dickinson, and Kaleem Siddiqi. DeepFlux for Skeletons in the Wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5287–5296, 2019.

[26] Changchang Wu, Jan-Michael Frahm, and Marc Pollefeys. Detecting Large Repetitive Structures with Salient Boundaries. In *European conference on computer vision*, pages 142–155. Springer, 2010.